アスキー・マイクロソフトFE 監修

MSX2 Technical Hand Book

MSX2 [illegible]・ハンドブック

アスキー出版局

アスキー・マイクロソフトFE 監修

MSX2 Technical Hand Book

MSX2テクニカル・ハンドブック

アスキー出版局

カバーデザイン●スタルカ

はじめに

MSXパーソナルコンピュータ・システムが世に現れてから，すでに2年以上の月日が経過しました．しかし残念ながらMSXがその本来の目標である“ホームコンピュータ・システム”として世間に認められるには，今少しの時間が必要なようです．それどころか，MSXは当初の意図とはかなり掛け離れた部分で評価されることも多く，“単なるゲームマシンにすぎない”という誤った認識さえ一般に浸透しつつあります．これはなぜでしょう？

MSXはソフトウェア的には現在の8ビットマシンの最先端をいくものです．BASICにおける数値の計算は有効桁数14桁の10進演算を基本とし，10兆の単位まで誤差なく表現できます．これはビジネスユースにも十分対応可能です．また実行速度の面でも，MSXのBASICは8ビットマシンの中では最高速の部類に入ります．ディスク操作の環境としてはMS-DOS ver 1.25をもとに作成されたMSX-DOSが用意されています．このDOSは，MS-DOSと同様の優れた操作性を備えるとともに，システムコールレベルでCP/M-80と互換性を持ち，CP/M上で作成されたプログラムであれば，簡単にMSX-DOSに移植することができます．BASICやDOSの持つ基本的な入出力操作は，BIOSあるいはシステムコールとして公開され，ユーザーが独自に利用することも可能となっています．

これらの機能にもかかわらず，MSXはその真価が実に目に止まりにくいマシンでした．大きな理由の1つは，MSXの画面表示能力にあったといっても間違いはないでしょう．実際，少なくとも80文字×25行のテキスト，640ドット×200ドットのグラフィックが標準となった現在のパソコンの中で，MSXの画面表示の機能は確かに見劣りするものでした．

そのような状況下、1985年5月にMSX2システムの仕様が発表されました．このMSX2は，従来の仕様と完全な互換性を保ちつつ多くの新しい機能が追加されています．特に画面表示の機能には大きな改良が加えられました．その結果，いまやMSXはグラフィック機能の面でも8ビットマシンの最高峰に位置するものとなったのです．唯一のネックであった画面表示機能が改善された今，MSXはいよいよその能力を多くの分野で発揮していくことでしょう．本書ではこのMSX2の構造と機能を詳しく解説しました．MSX2のグラフィック機能，MSX-DOSの機能，そしてBIOSルーチンの使用法など，MSX2に関する資料やデータは可能な限り掲載しています．これらの情報をMSX2システムの理解とプログラムの作成に役立たせていただければ幸いです．なお実際には“MSX1”という呼称は存在しませんが，本書の中では従来の仕様に沿って作られたMSXをMSX1と呼び，MSX2との区別をしています．

本書はソフトウェアハウスのプログラマなど，ある程度パソコン使用の経験をつんだ方を主な読者対象としています．限られたページ数の中に多くのデータを盛り込むため，初歩的な事項についての説明は省かざるを得なかったのです．まだパソコンに触れてまもない初心者の方は，今後アスキー書籍編集部から発行されるMSX2の入門書にぜひご期待ください．

本書を読む前に

第1部　MSX2システム概要

MSX 2の機能をハード／ソフトの両面から解説します．

第2部　BASIC

MSX 2の拡張部分をサポートするために強化されたMSX-BASIC ver 2.0の機能を述べ，さらにその内部構造，マシン語とのリンク方法などを解説します．

第3部　MSX-DOS

MSX-DOSの操作とシステムコールの使用法を解説します．

第4部　VDPと画面表示

MSX 2の画面表示機能を担う新VDP(MSX-VIDEO)の機能と使用法を解説します．

第5部　BIOSによる周辺装置のアクセス

MSXではすべての周辺装置はBIOSを通してアクセスします．ここでは，その具体的な使用法を解説します．

APPENDIX

BIOS一覧，ワークエリア一覧に加え，BASICで使用している浮動小数点演算パッケージ(Math-Pack)，画像データ転送ルーチン(ビットブロック・トランスファ)の両者を公開しています．

BIOSエントリおよびワークエリアの表記法について

● MSX 2ではBIOSがMAIN-ROMとSUB-ROMに分割されて存在します．これを区別するため，本書では次のような記法でBIOSエントリを表示しています．

BIOSエントリの表示例

- KEYINT（0038H／MAIN）← MAIN-ROMの0038H番地
- WRTCLK（01F9H／SUB）← SUB-ROMの01F9H番地

● また，ワークエリアの表示には，次の記法を用いて先頭アドレスと使用するバイト数を表しています．

ワークエリアの表示例

- VARTYP（F663H, 1）← F663H番地の1バイトを使用
- BUF（F55EH, 258）　← F55EH番地以降258バイトを使用

目　次

第5部　BIOSによる周辺装置のアクセス　263

第1部

MSX 2 システム概要

MSX1からMSX2へのバージョンアップは，MSX1との完全互換性を保つことを大前提に行われていますが，その拡張された機能は画面関係を筆頭にかなりの数，かなりの規模に上っています．そこでこの第１部では，MSX1からMSX2になって向上した機能を中心に，ブロック図や規格表などをまじえて簡単に紹介していくことにします．ここでの知識は概念的なものではありますが，第２部以降の解説を理解するためには必要なものです．

1章 MSX1からMSX2へ

まず最初に，MSX というコンピュータがそもそもどのような主旨で企画・製品化されたものかを振り返り，そのあと MSX 1 から MSX 2 への変化をまとめます．

1.1 MSX とは

MSX は，1983 年秋に 8 ビットパソコンの統一仕様として発表されました．発足当時は“互換性”という言葉が正しく理解されていないことも多く，MSX ならほかのどんなパソコンのソフトも動くというような誤解もあったようです．また逆に，MSX は MSX のソフトしか動かないのなら，PC シリーズは PC 用，FM シリーズは FM 用のソフトしか動かないのとどこが違うのかという意見も聞かれました．

コンピュータというマシンがパソコンという形で，一般の人々にも手が届くものになったのはまだここ数年のことです．出始めの頃は一般といっても，パソコンを買ったのは本当のマニアばかりで，互換性はおろか，実用に供することさえ難しかったものでした．ただひたすらコンピュータというものに触れ，その勉強ができればよかったのです．しかし，今では普及率もかなりのものとなり，意外な人がなんとなくパソコンを持っているというようなことにさえなってきました．つまり，パソコンはテレビやラジカセのような家電製品になろうとしているわけです．こうなってくると“互換性”という問題が前面にでてきます．メーカーごとに違った放送局が必要なテレビや違ったテープが必要なラジカセといったものは，果たして普及するでしょうか？ 家電製品としてのコンピュータのソフトは絶対に互換性を持っていなければならないのです．

こうしたことを見越した上で始まったのが MSX です．しかし，コンピュータのメリットが“何でも使える”ことだということを考えると，仕様を統一するといっても，あまりに決めるべき事項が多すぎ，またハードウェアの進歩が速過ぎるために，“決定的”な仕様などは存在し得ないということになります．したがって，MSX では最も基本的なハードウェアおよび DOS，BASIC といった基本ソフトウェア，あらゆる拡張の基礎となるスロットの仕様を決めることから始まりました．コンピュータが単体で利用されているうち(ゲームなど)はまだよいのですが，これが様々な“周辺機器”とつながり，様々なデータを処理，蓄積するようになっていくと，後から後から決めなければならないことが増えてきます．幸い，MSX には多くの家電メーカーの賛同が得ら

れ，ありとあらゆる家電製品をコンピュータとつなげる試みがなされ，仕様ができあがっていきました．これは，MSXを“核”として皆が集まってくるのであって，あるひとつのコンピュータのメーカーが自社のマシンを中心にして，他の“周辺機器”をそろえていくのとは根本的に発想が違うのです．

MSXはこのようにして進められてきたもので，その根底は非常にしっかりとしたものです．BASICの基本演算に倍精度のBCDを採用し，ディスクシステムとしてMS-DOSと同一のファイルフォーマットを使っていることなどは，MSXが本格的コンピュータとして設計されていることを示す良い例です．今後は，こうしたMSXの底力がいろいろな分野で発揮されるようになることでしょう．

1.2　MSXの環境と拡張性

MSXは1985年12月に，普及台数100万台を超え，主にゲーム機または入門機として小中学生の間で使われています．しかし，次第に通信端末，日本語ワープロ，ファクトリ・オートメーション，オーディオ・ビジュアル・コントロールといった分野での利用も進み，その真価を発揮しつつあります．開発環境としてもディスク・システム，MSX-DOSの整備が進み，Cコンパイラ，FORTH，LOGOなどの高級言語が次々と発表，発売されています．BASIC ROM内の入出力ルーチンの集合体であるBIOSや，ディスク・インターフェイスROM内のBDOS(CP／Mのシステムコールとほぼ互換性を持ち，さらに大きく改良されている)は公開され，優れたプログラミング環境を提供しています．漢字の扱い方や，マウス，ライトペン，RS-232Cの規格なども標準化され，周辺機器の統一も進んでいます．また国際的なレベルでもキーボードやキャラクタセットの一部を除いて同一のものとなっていて，それらの違いも各国ごとに規定されています．

周辺機器としては，プリンタ，ディスク・ドライブ，マウスなどの標準的なものの他，LD，VTR，シンセサイザのコントローラ，画像取り込みなどのAV関係の装置，ロボット・コントロール，室温管理などに代表されるFA用の機器，モデムや電話回線を利用するための様々なアダプタ，血圧計と組み合わせた健康管理機器などが開発されています．コンピュータの利用方法がいかに広がってゆくかを見せつけられる思いです．

ゲーム以外の一般的なアプリケーション・ソフトウェアも，最近ではディスク対応のものが増え本格的な実用性を備えてきています．例をあげると，文節変換のできる日本語ワープロ，より上位のシステムとデータを共有できるようなデータベース，CAI，CADなどのシステムがすでに発売されています．

1.3　MSX 2 での拡張内容

MSX 2 は，1985 年 5 月に MSX の上位互換性を持ったシステムとして発表されました．MSX の仕様に沿って作成されたプログラムは，マシン語のものであってもなんらの変更を加えることなく MSX 2 上で動作します．また，カセットやディスク上に蓄積されたデータやプログラムもそのまま使うことが可能です．MSX 2 において拡張されたのは主に画面関係の機能で，解像度，扱える色，グラフィック処理の速度などが大幅に改善されました．そのほかにもバッテリバックアップされた時計機能，標準で 64K に統一されたメイン RAM を使った RAM ディスク機能などがあげられます．以下本書では，この MSX 2 仕様に沿って作られたコンピュータを MSX 2 と呼び，従来の MSX 仕様に沿って作られたコンピュータを MSX 1 と呼ぶことにします．

まず，MSX 1 と MSX 2 のハードウェアではどこが違うかを，**図 1.1** と**図 1.2** のシステム構成図，および**表 1.1** に示します．以下この中で拡張された各項目について，順に見ていきましょう．

		MSX2仕様	MSX1仕様
CPU		Z80A 相当品(クロック 3.579545MHz±1%)	
メモリ	ROM	48K(MSX-BASIC ver2.0) MAIN-ROM 32K SUB-ROM 16K	32K(MSX-BASIC ver1.0) MAIN-ROM 32K
	RAM	64K以上	8 K以上
	VRAM	64Kまたは128K	16K
VDP 使用LSI		V-9938(MSX-VIDEO)	TMS9918相当品
CMT		FSK方式 1200／2400 baud	
PSG		8オクターブ3重和音出力(AY-3-8910相当品)	
キーボード(日本版)		英数，ひらがな，カタカナ グラフィック記号対応 JIS配列／50音配列対応 (ローマ字入力可能)	英数，ひらがな，カタカナ グラフィック記号対応 JIS配列／50音配列対応
フロッピーディスク※		フォーマットはMS-DOSに準拠	
プリンタ		8ビットパラレル	(※)
ROMカートリッジ		I／Oバス ゲームカートリッジ，拡張バスに対応するスロットを持つ	
ジョイスティック		2	1または2(※)
漢字機能※		統一仕様	各社対応
CLOCK-IC		あり	なし
RAMディスク機能		あり	なし

※はオプションを表す．

表1.1　MSX2とMSX1の仕様比較

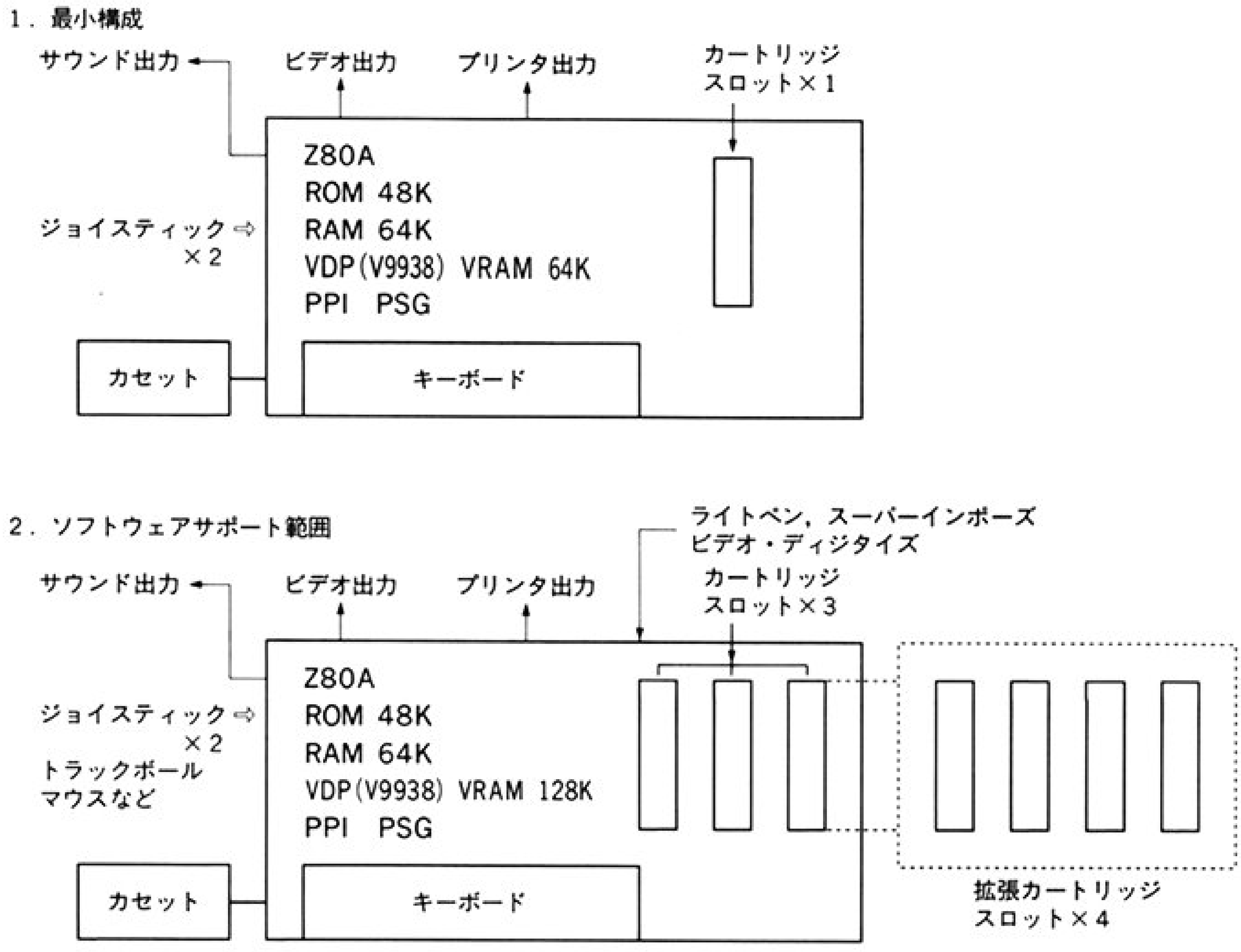


図1.1　MSX2システム構成

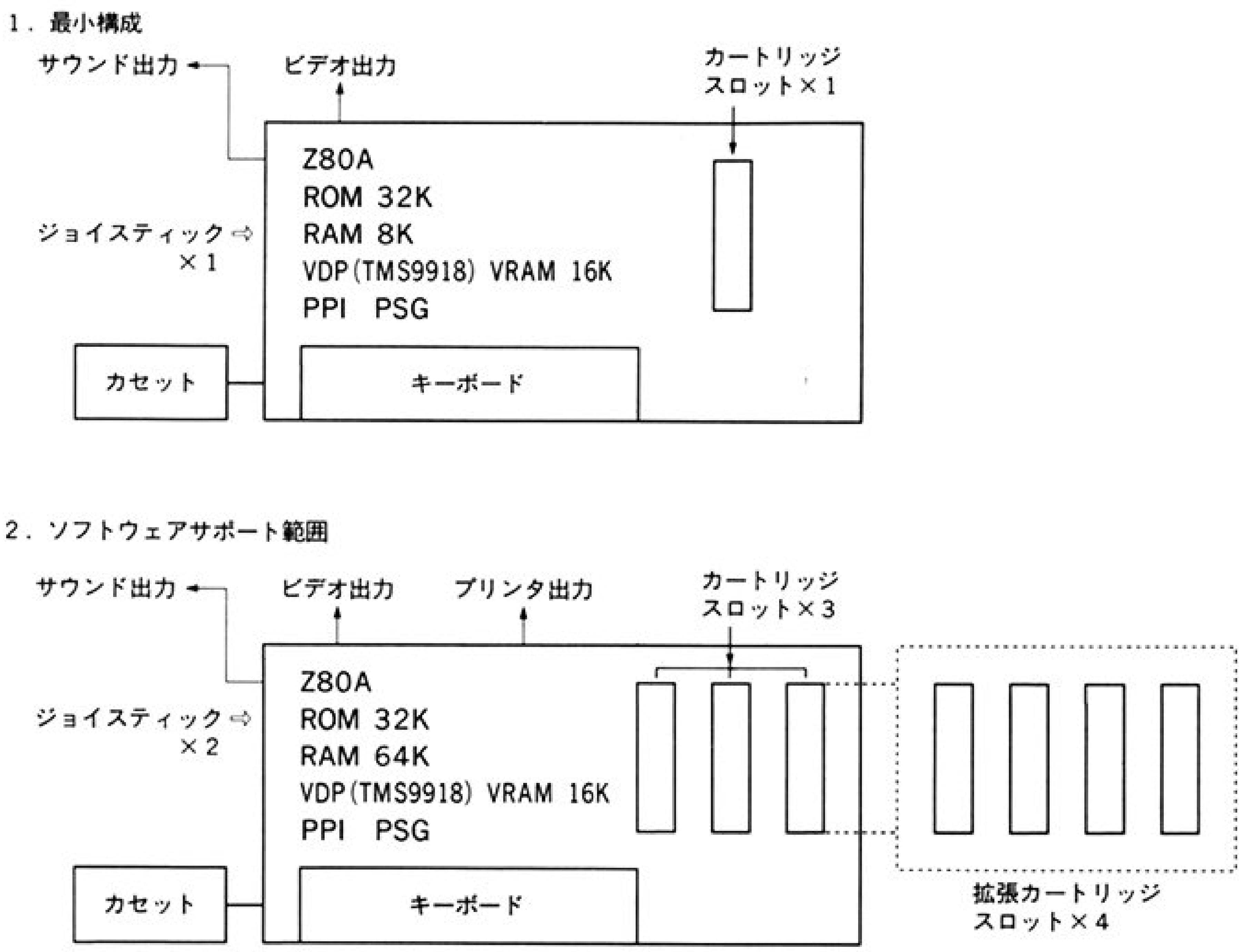


図1.2　MSX1システム構成

● MSX-BASIC

新 VDP，バックアップRAM，CLOCK-ICなどをサポートするため，BASICもver1.0からver 2.0へ大幅に拡張されました．もちろん，MSX 1との互換性は保たれています．ただし，新しく拡張された画面モードなどを使用する場合，今までとは指定方法や指定範囲が若干異なる場合がありますので注意が必要です．

●メモリ(ROM，RAM，VRAM)

MSX 2のメモリには，ROM，RAM，VRAMの3種類がありますので，それぞれについて説明しましょう．

ROM

ROMは標準で48K用意されています．MSX 1は，標準で32Kですから16K分増えたことになります．この増えた16K分のROMには，主にMSX 2に対して拡張された部分をサポートするプログラムが入っています．

32K側のROMにはBASICインタープリタのメインプログラムが入っているので“MAIN-ROM(メインロム)”，16K側のROMには拡張のためのプログラムが入っているので“拡張ROM”，あるいは“SUB-ROM(サブロム)”と呼んでいます．

RAM

RAMは標準で64K用意されており，拡張なしでMSX-DOSを動かすことができます．MSX1では8Kから64KまでRAM容量が様々で，大きなプログラムはRAMを増設しないと動かないことがありましたが，MSX 2ではそのようなことがなくなっています．

VRAM

画面表示機能強化のためVRAMは最低でも64Kと規定されています．MSX 1が16Kですから，実に4倍にも拡張されたことになります．しかし，現実的には最大実装の128Kのマシンが多く，そうなるとMSX 1の8倍にもなるのです．128K実装しているマシンでは，256色同時表示が可能です．

なお，VRAMが64K実装されているMSXのうち128Kに増設できない機種は，カタログまたは箱にVRAM64Kなどと表示されているはずです．

● VDP

MSXの画面の制御に使用されているのが“VDP(ビデオ・ディスプレイ・プロセッサ)”というLSIです．MSX 1で使われていたVDPはTMS9918でしたが，MSX 2にはV9938 (MSX-

VIDEO）が使われています．V9938は，TMS9918と完全に上位互換性を持ったチップで，TMS9918用に作られたソフトウェアはすべて変更なしに動作します．

表1.2にVDPの仕様を，**表1.3**にV9938の各画面モードを示します．V9938はこのほかにも，ディジタイズ機能，スーパーインポーズ機能，ハードウェア・スクロール機能などを持った優れたLSIなので，第4部でさらに詳しく解説しています．

		V9938	TMS9918
スクリーンモード		10種類（表1.3参照）	4種類
ドット数（横×縦）		最大512×212 インターレース機能により 縦424ドットも可能	最大256×192
色	指定色	最大512色	最大16色
	同時表示色	最大256色	最大16色
文字構成		英数文字＋ひらがな＋カタカナ＋グラフィック記号 256種8×8ドット	
スプライトの色		1枚につき最大16色	1枚につき1色
パレット機能		あり	なし

表1.2　VDPの仕様

モード	文字数	ドット	色	パレット	スプライト
*テキスト1	40×24		512色中2色	あり	なし
テキスト2	80×24		512色中4色	あり	なし
*マルチカラー		64×48	512色中16色	あり	モード1
*グラフィック1	32×24		512色中16色	あり	モード1
*グラフィック2		256×192	512色中16色	あり	モード1
グラフィック3		256×192	512色中16色	あり	モード2
グラフィック4		256×212	512色中16色	あり	モード2
グラフィック5		512×212	512色中4色	あり	モード2
グラフィック6		512×212	512色中16色	あり	モード2
グラフィック7		256×212	256色中256色	なし	モード2

*はTMS9918と同機能のモードです．ただしパレットはV9938のみの機能です．

表1.3　V9938画面モード一覧

●バッテリバックアップのCLOCK-IC

MSX 2には電池によりバックアップされているRAMがI／Oポートより接続されており，リセット時のセットアップ情報の保存と，カレンダ機能のために使用されています．セットアップ情報とは，リセット時の画面の色や画面モードなどを指し，これを保存しておくことによって，

起動時にユーザーが望む環境に設定することができます．

CLOCK-ICはMSXの電源とは完全に独立して動いているので，リセット後，新たに時間を設定する必要がなくなりました．

● RAMディスク機能

64KバイトのRAMを実装しているMSX 1のマシンでBASICを使用している場合，32KのRAMが使用されるのみで，残り32Kはまったく使われませんでした．MSX 2では，この眠っているRAMをRAMディスクとして使用することができます．ディスク・ドライブを持っていない人がBASICプログラムを一時的にロード，セーブするには，とても便利な機能です．

● ローマ字カナ変換機能

MSXのかなキー配列には，JIS配列と50音配列の2種類がありますが，英字入力に慣れ親しんだ人にとってはどちらの配列であっても，かな文字の入力は容易ではないでしょう．このため，MSX 2ではローマ字によるひらがなとカタカナの入力が可能となっており，マン・マシンインターフェイスが一段とよくなりました．このローマ字かな変換機能は，MSX-DOS，MSX-BASICどちらのモードでも利用できます．

2章　MSX 2 システム概要

ここでは，MSX 2 システムの概要を，ソフトウェア／ハードウェアの両面から簡単に見ていくことにします．なお，概念的理解の妨げにならないように，実際のソフトウェア開発時に必要になるような具体的内容(VRAM マップ，I／O マップ，インターフェイス規格など)は，なるべく巻末の APPENDIX にまとめて掲載することにしました．

2.1　ハードウェア概要

まず，MSX 2 の全体のハードウェア構成を理解していただくために，ブロック・ダイヤグラムを図1.3に示します．

2.1.1　アドレスマップ

●メモリマップ

MSX2 に標準で装備されているメモリは，MAIN-ROM，SUB-ROM，64KRAM の 3 つです．それらのメモリは，物理的にはそれぞれ独立した 64K のアドレス空間にあり図1.4の(1)のように配置されています(64K の各空間を“スロット”，それを 16K ずつ 4 つに分けた各エリアを“ページ”という)．(2)，(3)はそれぞれ BASIC，MSX-DOS を利用する時のメモリの使われ方です．

では,それぞれのメモリについて(イ)のメモリマップを図1.5に,(ロ)のメモリマップを図1.6に，(ハ)のメモリマップを図1.7に示します．

メモリマップ以外で，仕様が規定されているものに，VRAM マップ，I／O マップがありますが，それらは巻末の APPENDIX にまとめて掲載しています．

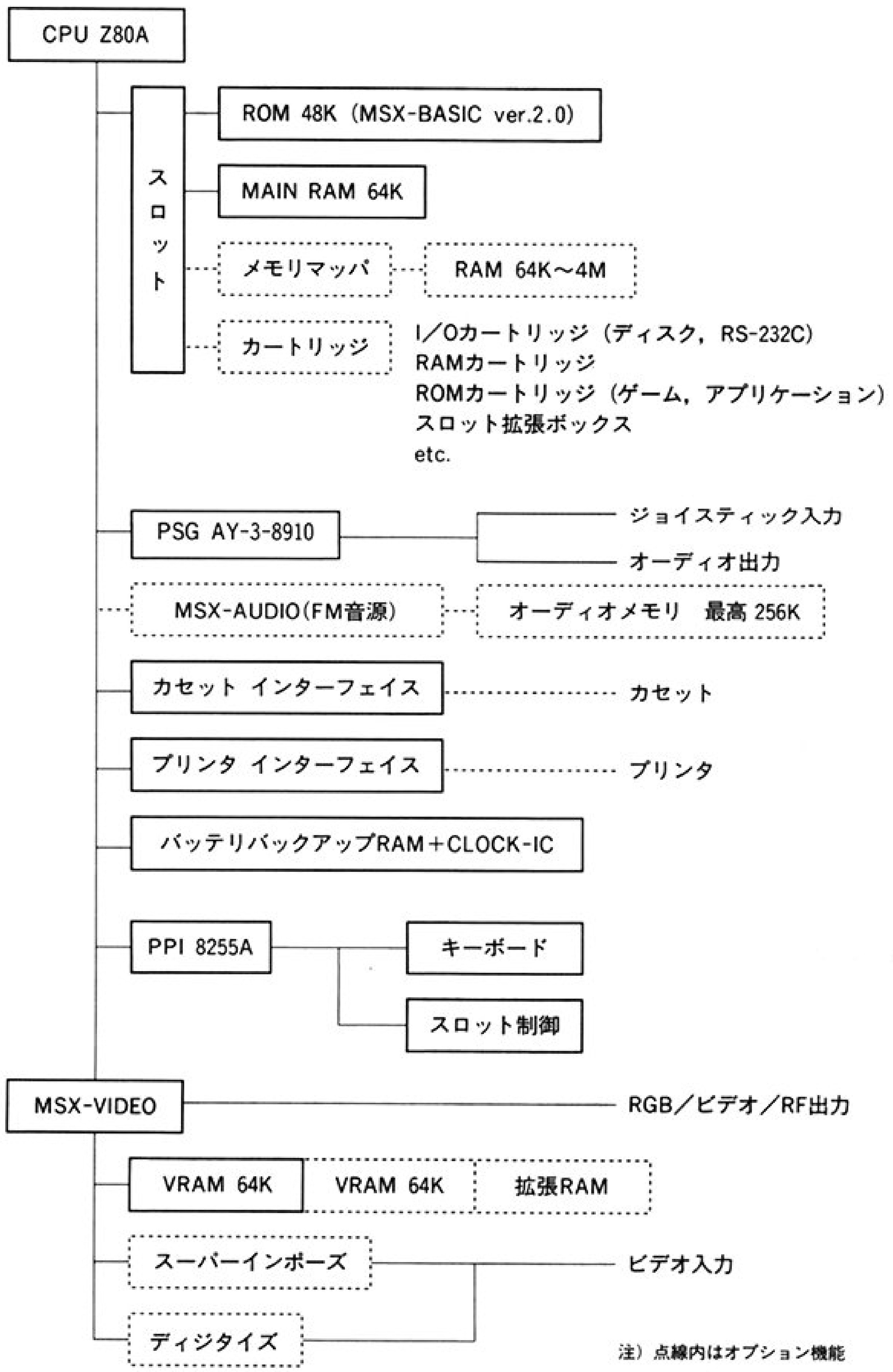
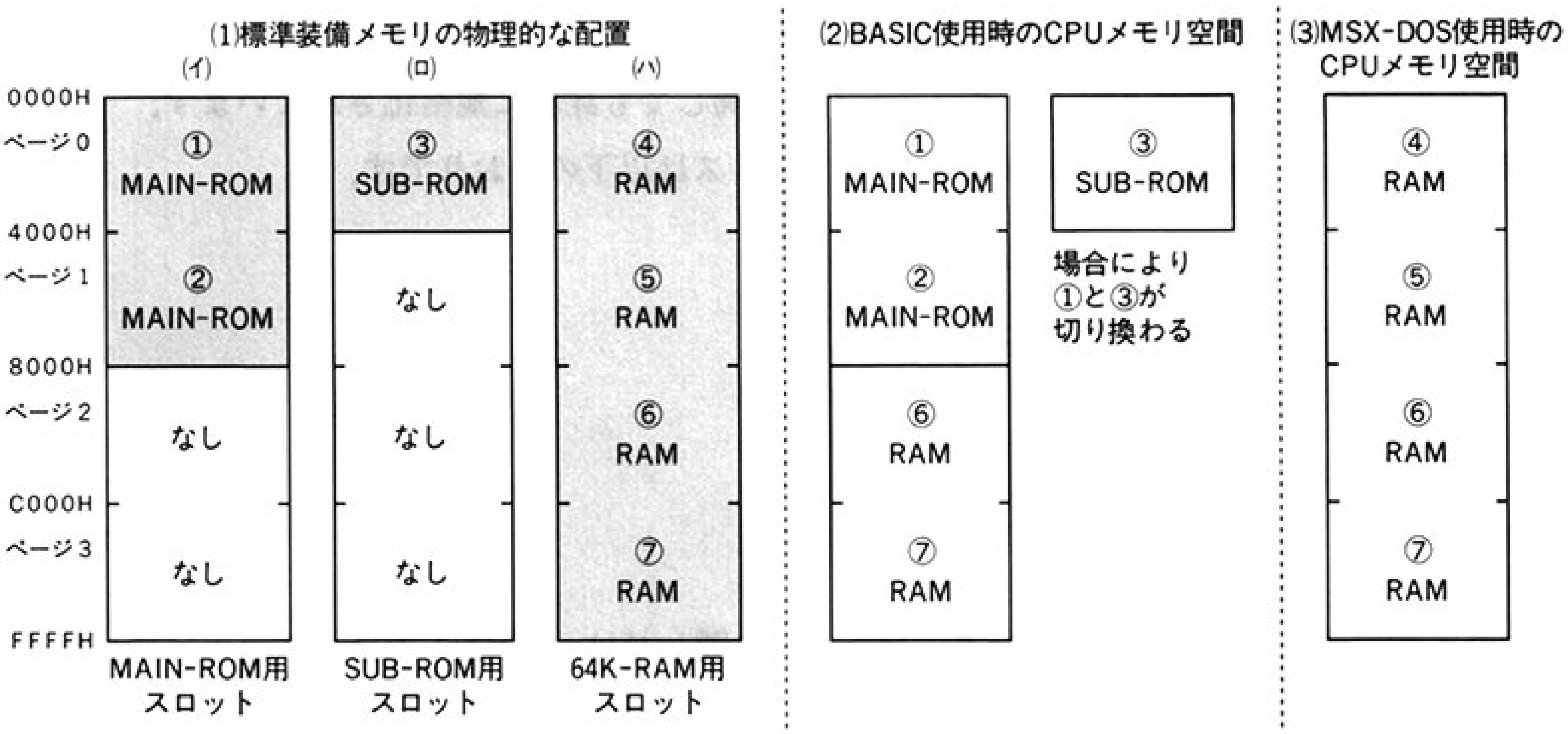


図1.3　MSX2のブロックダイアグラム



注）64K-RAMの4つのページ（④～⑦）は，実際は同じスロットにあるとは限りません

図1.4　MSX2の標準装備メモリ



図1.5　MAIN-ROMのメモリマップ

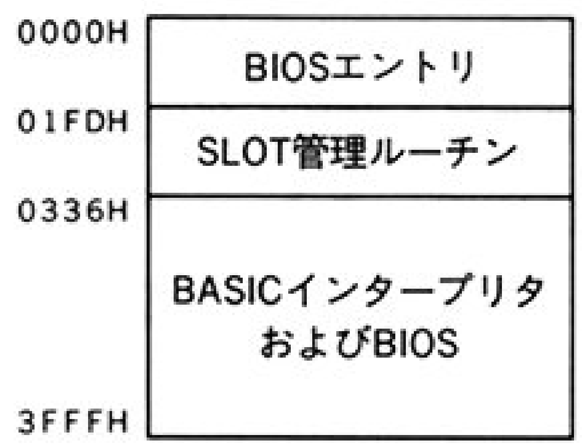


図1.6　SUB-ROMのメモリマップ

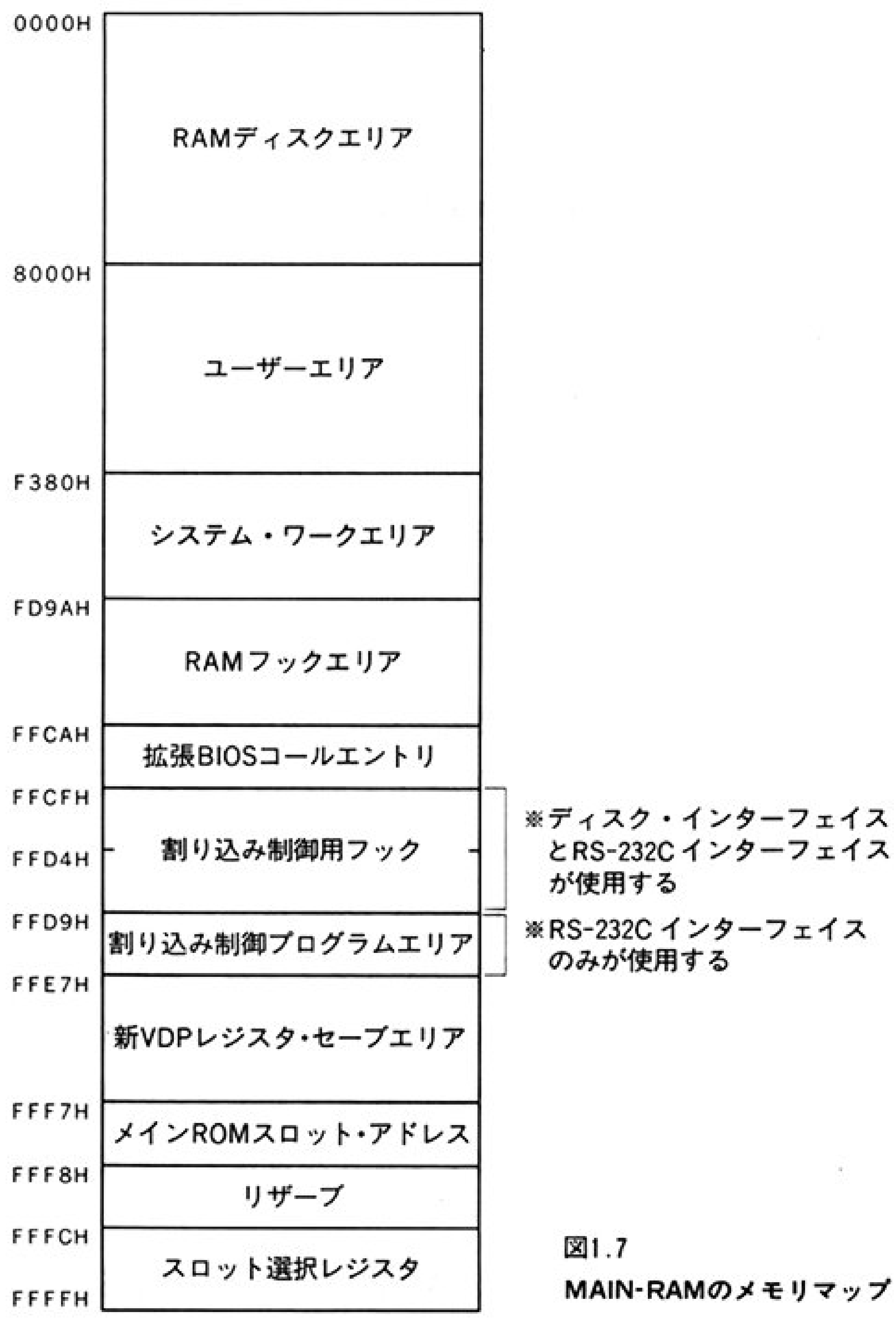


図1.7
MAIN-RAMのメモリマップ

2.1.2 周辺機器とのインターフェイス

MSX 2では，周辺機器とのインターフェイスに関しても詳細に規格化されています．
標準仕様として規格化されているインターフェイスは以下のとおりです．

・ディスプレイ・インターフェイス
・オーディオ・インターフェイス
・カセット・インターフェイス
・汎用入出力インターフェイス
・プリンタ・インターフェイス

このうち，最後のプリンタ・インターフェイスに関しては，MSX 1ではオプションでしたが，MSX 2から標準仕様に加えられています．

ディスク・ドライブのインターフェイスは，MSX 1と同様オプションになっていますが，MSX 2のメインRAMが標準で64Kであることを考えると，標準仕様に近い存在といえるでしょう．

その他，MSXの大きな特徴であるカートリッジの仕様など詳細な情報はAPPENDIXに掲載しています．

2.2 ソフトウェア概要

MSXにはBASICモードとDOSモードの2つのソフトウェア環境が用意されています．BASICモードではMSX-BASICによるプログラムの開発，実行といったことが簡単に行えるようになっており，通常はこのモードが広く使用されています．パソコンが現在これだけの普及率を得ているのも，ひとつにはこのBASIC言語の"誰にでも使える"という点が評価された大きな要因であるといえます．

DOSモードではMSX-DOSを核としてさまざまな言語，ユーティリティあるいはアプリケーションといったものを使用することができるようになっています．一般にDOSとは，ハードウェアの違いを補い，そのDOS上であればマシンを選ぶことなく同一のプログラムが実行できる場を提供するもので，これにより今までに蓄積されてきたソフトウェア資源を有効に利用できることになります．もちろんMSX-DOSにもその精神はおおいに受け継がれています．まず，ディスクのフォーマットが16ビットマシンでは一般的なMS-DOSと同一であるため，MS-DOS上で作成されたC言語などのソースプログラムをMSX-DOS上でコンパイルし実行するといったことが簡単にできます．また，8ビットマシンの世界で膨大なアプリケーションの蓄積を持つCP/M上のソフトウェアが，ファイル変換をするだけで，ほとんどそのまま実行できることも見逃せません．

そして特筆すべきなのは，MSXにおいてはBASICとDOSが同一のディスクフォーマットを

持ち，資源が共有できるという点でしょう．両者は**図1.8**に示すように，BIOS(Basic I／O System)を共通の基盤として持つ均一化されたソフトウェア環境の上に成立しています．ディスク操作の基本となるBDOS(Basic Disk Operating System)もまた，このBIOS上に構築されているのです．MSXでは，この共通のBDOSとBIOSを通して，BASICとDOSに統一されたプログラミング環境を提供しています．

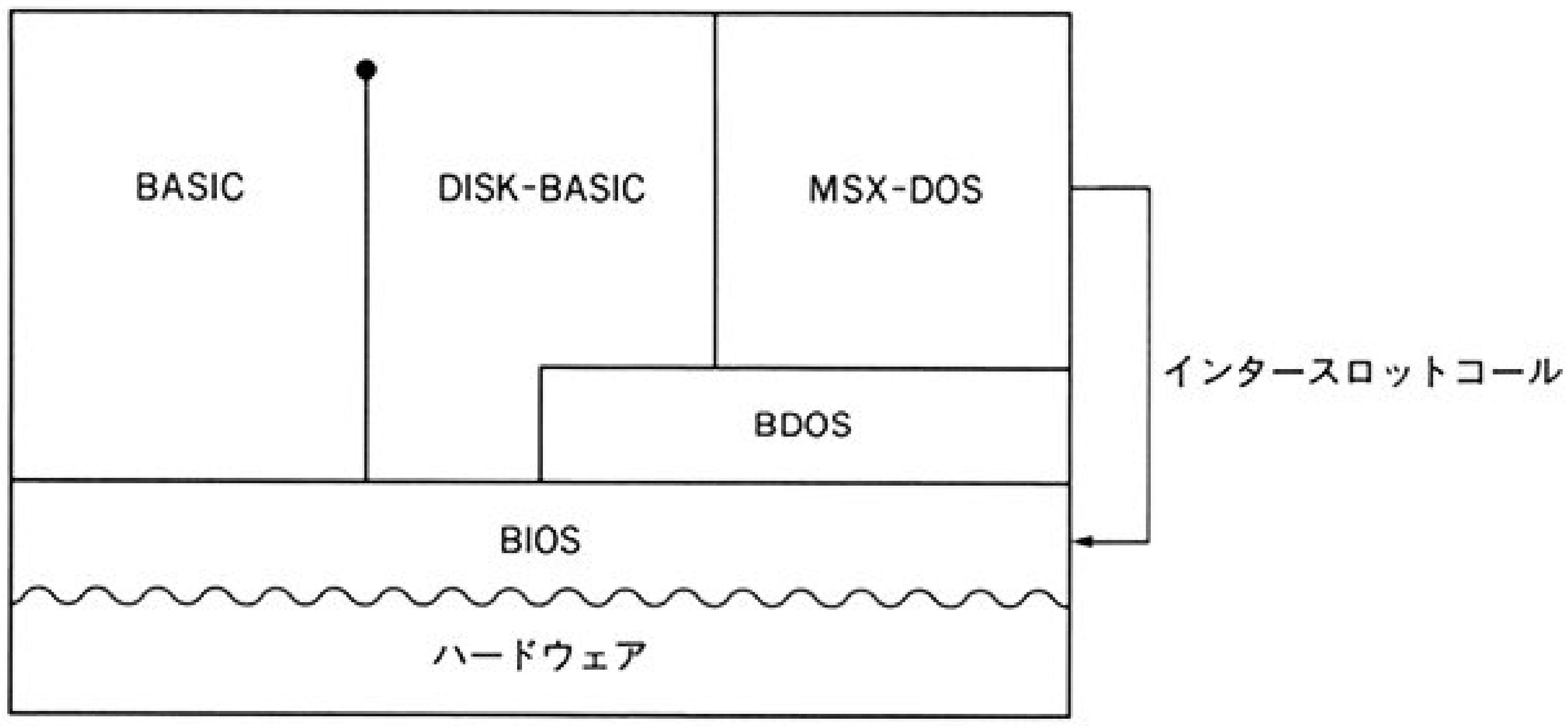


図1.8　MSX1およびMSX2のソフトウェア階層図

第2部

BASIC

優れたハードウェア機能を活かすため，MSX 2ではBASICもバージョンアップされました．これがMSX BASIC ver 2.0です．また，ディスクシステムを使用している場合には，このver2.0にさらにディスク操作の命令が追加されたMSX DISK-BASICが利用できます．本章では，この2つのBASICについて説明します．

1章　命令一覧

まず，BASICの各命令について，その構文と機能の一覧を示します．それぞれの命令は**図2.1**の書式で解説されています．

命令の書式	
命令のタイプ	命令の機能，動作

図2.1　命令一覧表の書式

(a) 命令の構文

キーワードの前に“＊”が付いている命令は，ver 1.0の命令に構文や機能の上で変更があったもの，またはver 2.0で新しく追加されたものです．

構文の記述には，以下の記法を用いています．

- **[項目]** ……………………省略可能であることを示す．
- **[，項目・・・]** …………同様な項目をいくつでも並べてよいことを示す．
- **{項目1¦項目2}** ………どれかひとつの項目を選ぶことを示す．

また，構文の中で使用されている<ファイル名>とは，以下のような形式で，入出力の対象となるI／O機器やファイルを指定する文字列です．この中で，カセットファイルに対する< *file name* >は6文字以下の任意文字列，ディスクおよびRAMディスクに対する< *filename* >は，“< *filename* 前部（8文字以下）>＋<拡張子（3文字以下）>”の形式の文字列で表されます．<ドライブ>はA～H（上限は接続されているドライブの数により決定）の1文字で表されます．

“CAS：< *filename* >”………………… カセットファイル
“MEM：< *filename* >”……………… RAMディスク
“CRT：”…………………………………… テキスト画面
“GRP：”…………………………………… グラフィック画面
“LPT：”…………………………………… プリンタ
“<ドライブ>：< *filename* >”……… ディスクファイル

(b) 命令のタイプ

以下の 4 種類があります.

- **ファンクション** …………与えたパラメータにしたがって，なんらかの値を返すもの
- **システム変数**……………BASICシステムが持っている変数．一般に代入も可能
- **ステートメント**…………なんらかの動作を行うもの
- **コマンド**…………………BASICインタープリタ自身に関する指示を与えるもの

(c) 命令の機能または動作

各命令がどのような動作を行うものか簡単に述べます．ver 2.0 で追加または変更された命令については，2 章でさらに詳しい説明を行います．

1.1　MSX BASIC ver 2.0 の命令

A

ABS (<数式>)	
ファンクション	<数式>の絶対値を返す．
ASC (<文字列>)	
ファンクション	<文字列>の最初の文字のキャラクタコードを返す．
ATN (<数式>)	
ファンクション	<数式>の逆正接（アークタンジェント）を返す．角度の単位はラジアン．
AUTO [<行番号>[, <増分>]]	
コマンド	行番号を自動的に発生する．

B

* BASE (<数式>)	
システム変数	VRAM 上に割り当てられている画面関係のテーブルアドレスを持つ．
BEEP	
ステートメント	BEEP 音を音声端子に出力する．
BIN $ (<数式>)	
ファンクション	<数式>の値を 2 進表記の文字列に変換し，その結果を返す．

BLOAD　"<ファイル名>" [, R [, <オフセット値>]]
コマンド　マシン語プログラムをロードする.

BSAVE　"<ファイル名>", <開始アドレス>, <終了アドレス>
[, <実行開始アドレス>]
コマンド　マシン語プログラムをセーブする.

C

CALL　<拡張ステートメント名> [(<引数> [, <引数>・・・])]
ステートメント　各種のカートリッジを差し込むことによって追加された拡張ステートメントを呼び出す.

*** CALL MEMINI [(<RAM ディスクの上限>)]**
ステートメント　RAM ディスクで使うメモリの上限を指定する.

*** CALL MFILES**
ステートメント　RAM ディスクに記録されているファイル名を表示する.

*** CALL MKILL ("<ファイル名>")**
ステートメント　RAM ディスク中のファイルを削除する.

*** CALL MNAME ("<旧ファイル名>" AS "<新ファイル名>")**
ステートメント　RAM ディスク中のファイル名を付け換える.

CDBL (<数式>)
ファンクション　<数式>の値を倍精度実数値に変換し, その結果を返す.

CHR$ (<数式>)
ファンクション　<数式>の値のキャラクタコードを持つ文字を返す.

CINT (<数式>)
ファンクション　<数式>の値を整数値に変換し, その結果を返す.

*** CIRCLE　{(X, Y) | STEP (X, Y)}, <半径> [, <色> [, <開始角度> [, <終了角度>**
[, <比率>]]]]
ステートメント　(X, Y) を中心とし, <半径>で指定する大きさの円を描く.

CLEAR　[<文字列領域の大きさ> [, <メモリの上限>]]
ステートメント　変数の初期化およびメモリ領域の大きさの設定をする.

CLOAD　["<ファイル名>"]
コマンド　カセットからプログラムをロードする.

CLOAD?　["＜ファイル名＞"]
コマンド　メモリ上のプログラムとカセット上のプログラムを比較する．

CLOSE　[[#]＜ファイル番号＞[，[#]＜ファイル番号＞・・・]]
コマンド　＜ファイル番号＞に対応するファイルを閉じる．

CLS
ステートメント　画面を消去する．

*** COLOR　[＜前景色＞ [,＜背景色＞ [,＜周辺色＞]]]**
ステートメント　画面の各部の色を指定する．

*** COLOR　[=NEW]**
ステートメント　パレットを初期化する．

*** COLOR=(＜パレット番号＞,＜赤の輝度＞,＜緑の輝度＞,＜青の輝度＞)**
ステートメント　パレットに色を設定する．

*** COLOR=RESTORE**
ステートメント　カラーパレット保存テーブルの内容をパレット・レジスタに入れる．

*** COLOR SPRITE(＜スプライト面番号＞)=＜色＞**
ステートメント　＜スプライト面番号＞のスプライトを指定した色にする．

*** COLOR SPRITE$(＜スプライト面番号＞)=＜文字式＞**
ステートメント　スプライトの横ラインごとの色を＜文字式＞で指定する．

CONT
コマンド　停止したプログラムの実行を再開する．

*** COPY　＜ソース＞　TO　＜デスティネーション＞**
ステートメント　画面，配列，ディスクファイル間で画像データを転送する．

*** COPY SCREEN　[＜モード＞]**
ステートメント　カラーバスのデータをVRAMに書き込む（オプション機能）．

COS(＜数式＞)
ファンクション　＜数式＞の値に対する余弦（コサイン）を返す．角度の単位はラジアン．

CSAVE　"＜ファイル名＞"[,＜ボーレート＞]
コマンド　プログラムをカセットにセーブする．

CSNG（＜数式＞）
ファンクション　　＜数式＞の値を単精度実数値に変換し，その結果を返す.

CSRLIN
システム変数　　テキスト画面上のカーソルの行位置を持つ．代入はできない.

D

DATA　＜定数＞[,＜定数＞・・・]
ステートメント　　READ命令で読み出すためのデータを用意する.

DEF FN＜名前＞ [(＜引数＞[,＜引数＞・・・])]=＜関数の定義式＞
ステートメント　　ユーザー関数を定義する.

DEFINT　＜文字の範囲＞[,＜文字の範囲＞・・・]
ステートメント　　指定された変数を整数型として宣言する.

DEFSNG　＜文字の範囲＞ [,＜文字の範囲＞・・・]
ステートメント　　指定された変数を単精度実数型として宣言する.

DEFDBL　＜文字の範囲＞ [,＜文字の範囲＞・・・]
ステートメント　　指定された変数を倍精度実数型として宣言する.

DEFSTR　＜文字の範囲＞ [,＜文字の範囲＞・・・]
ステートメント　　指定された変数を文字型として宣言する.

DEF USR [＜番号＞]=＜開始アドレス＞
ステートメント　　USR関数が呼び出すマシン語ルーチンの実行開始アドレスを定義する.

DELETE　{[＜始点行番号＞−＜終点行番号＞] ¦ ＜行番号＞ ¦ −＜終点行番号＞}
コマンド　　プログラム中の指定された部分を削除する.

DIM　＜変数名＞（＜添字の最大値＞ [,＜添字の最大値＞・・・]）
ステートメント　　配列変数を定義し，メモリに割り当てる.

DRAW　＜文字式＞
ステートメント　　＜文字式（DRAWマクロ）＞にしたがって画面に線を描く.

E

END

ステートメント　　プログラムを終了し，ファイルをすべて閉じ，コマンドレベルに戻る．

EOF（＜ファイル番号＞）

ファンクション　　ファイルの終わりに達したかを調べ，終わりなら－1を返す．

ERASE　＜配列変数名＞［,＜配列変数名＞・・・］

ステートメント　　配列変数を削除する．

ERL

システム変数　　発生したエラーのエラーコードを持つ．代入はできない．

ERR

システム変数　　エラーが発生した行番号を持つ．代入はできない．

ERROR　＜エラーコード＞

ステートメント　　プログラムをエラー状態にする．

EXP（＜数式＞）

ファンクション　　自然対数の底eの＜数式＞乗の値を返す．

F

FIX（＜数式＞）

ファンクション　　＜数式＞の小数点以下を取り去った値を返す．

FOR　＜変数名＞＝＜初期値＞　TO　＜終値＞［STEP　＜増分＞］

ステートメント　　FOR命令からNEXT命令までを，指定回数繰り返し実行する．

FRE（{＜数式＞¦＜文字式＞}）

ファンクション　　未使用のユーザーエリア，または未使用の文字領域の大きさを返す．

G

＊GET DATE　＜文字変数名＞［, A］

ステートメント　　日付を文字変数に代入する．

＊GET TIME　＜文字変数名＞［, A］

ステートメント　　時刻を文字変数に代入する．

GOSUB　<行番号>
ステートメント　　<行番号>にあるサブルーチンを呼び出す.

GOTO　<行番号>
ステートメント　　<行番号>へジャンプする.

H

HEX$(<数式>)
ファンクション　　<数式>の値を16進表記の文字列に変換し，その結果を返す.

I

IF　<条件>　THEN {<文>|<行番号>} [ELSE {<文>|<行番号>}]
ステートメント　　条件判断を行う.<条件>≠0ならば真とみなす.

IF　<条件>　GOTO　<行番号> [ELSE {<文>|<行番号>}]
ステートメント　　条件判断を行う.<条件>≠0ならば真とみなす.

INKEY$
ファンクション　　キーが押されている場合はその文字を，それ以外は空文字列を返す.

INP(<ポート番号>)
ファンクション　　<ポート番号>で指定されたポートを読み取り，その結果を返す.

INPUT　["<プロンプト文>";]<変数名> [,<変数名>・・・]
ステートメント　　キーボードから入力されたデータを指定の変数に代入する.

INPUT　#<ファイル番号>,<変数名> [,<変数名>・・・]
ステートメント　　ファイルからデータを読み込み，指定の変数に代入する.

INPUT$(<文字数> [, [#]<ファイル番号>])
ファンクション　　キーボードまたはファイルから，指定された長さの文字列を読み込む.

INSTR([<数式>,]<文字式1>,<文字式2>)
ファンクション　　<文字式1>の左側から<文字式2>を探し始め，見つかればその位置を，そうでなければ0を返す.<数式>は探し始める文字の位置.

INT(<数式>)
ファンクション　　<数式>以下の最大の整数を返す.

INTERVAL　{ON ; OFF ; STOP}
ステートメント　　タイマ割り込みを許可，禁止，保留する．

K

KEY　<キー番号>, <文字列>
コマンド　　ファンクションキーを再定義する．

KEY LIST
コマンド　　ファンクションキーの内容を表示する．

KEY (<キー番号>) {ON ; OFF ; STOP}
ステートメント　　ファンクションキー割り込みを許可，禁止，保留する．

KEY　{ON ; OFF}
ステートメント　　ファンクションキーの内容を画面下方に表示するかどうかを指定する．

L

LEFT$(<文字式>, <数式>)
ファンクション　　<文字式>の左側から<数式>個分の文字列を取り出す．

LEN(<文字式>)
ファンクション　　<文字式>の文字数を返す．

[LET]　<変数名>=<式>
ステートメント　　変数に<式>の値を代入する．

*** LINE　[{(X1, Y1) ; STEP (X1, Y1)}]− {(X2, Y2) ; STEP (X2, Y2)} [, <色> [, {B ; BF} [, <ロジカルオペレーション>]]]**
ステートメント　　画面に直線または四角形を描く．

LINE INPUT　["<プロンプト文>" ;]<文字変数名>
ステートメント　　キーボードから入力された文字データ1行分すべてを文字型変数に代入する．

LINE INPUT #<ファイル番号>, <文字変数名>
ステートメント　　1行単位のデータをファイルから読み込み，文字変数に代入する．

LIST　[[<行番号>]− [<行番号>]]
コマンド　　メモリ上のプログラムを画面に表示する．

LLIST　[[<行番号>]−[<行番号>]]
コマンド　　メモリ上のプログラムをプリンタに出力する.

LOAD　"<ファイル名>"[, R]
コマンド　　アスキーセーブしてあるプログラムをロードする.

* LOCATE　[<X 座標> [, <Y 座標> [, <カーソルスイッチ>]]]
ステートメント　　テキスト画面でのカーソル位置を指定する.

LOG (<数式>)
ファンクション　　<数式>の自然対数を返す.

LPOS (<数式>)
システム変数　　プリンタのヘッドの位置を持つ. 代入はできない.

LPRINT　[<式> [{ ; ¦ ,}<式>・・・]]
ステートメント　　プリンタに文字や数値を出力する.

LPRINT USING　<書式> ; <式> [{ ; ¦ ,}<式>・・・]
ステートメント　　<書式>に基づいてプリンタに文字や数値を出力する.

M

MAXFILES=<ファイル数>
ステートメント　　オープン可能なファイルの数を設定する.

MERGE　"<ファイル名>"
コマンド　　メモリ上のプログラムとアスキーセーブされた(外部記憶装置の)プログラムを混合する.

MID$(<文字式>, <数式1> [, <数式2>])
ファンクション　　<文字式>の<数式1>番目から<数式2>個分の文字を取り出す.

MID$(<文字変数名>, <数式1> [, <数式2>])=<文字式>
ステートメント　　文字変数中の<数式1>番目から<数式2>個分の文字を<文字式>の最初から<数式2>個分の文字と置き換える.

MOTOR　[{ON ¦ OFF}]
ステートメント　　カセットのモータをON／OFFする.

N

NEW
コマンド　　メモリ上のプログラムを削除し，変数をクリアする．

NEXT　[<変数名> [, <変数名>・・・]]
ステートメント　　FOR 命令の終わりを示す．

O

OCT$(<数式>)
ファンクション　　<数式>の値を 8 進表記の文字列に変換し，その結果を返す．

ON ERROR GOTO　<行番号>
ステートメント　　エラー処理ルーチンの開始行を定義する．

ON　<数式>　GOSUB　<行番号> [, <行番号>・・・]
ステートメント　　<数式>の値に応じた<行番号>のサブルーチンを実行する．

ON　<数式>　GOTO　<行番号> [, <行番号>・・・]
ステートメント　　<数式>の値に応じた<行番号>へジャンプする．

ON INTERVAL=<時間>　GOSUB　<行番号>
ステートメント　　タイマ割り込みの間隔と，割り込み処理ルーチンの開始行を定義する．

ON KEY GOSUB　<行番号> [, <行番号>・・・]
ステートメント　　ファンクションキーによる割り込み処理ルーチンの開始行を定義する．

ON SPRITE GOSUB　<行番号>
ステートメント　　スプライトの重なりによる割り込み処理ルーチンの開始行を定義する．

ON STOP GOSUB　<行番号>
ステートメント　　CTRL+STOP による割り込み処理ルーチンの開始行を定義する．

ON STRIG GOSUB　<行番号> [, <行番号>・・・]
ステートメント　　トリガボタンによる割り込み処理ルーチンの開始行を定義する．

OPEN　"<ファイル名>" [FOR　<モード>]　AS　#<ファイル番号>
ステートメント　　ファイルを指定したモードでオープンする．

OUT　<ポート番号>, <数式>
ステートメント　　<ポート番号>で指定された出力ポートにデータを送る．

P

* PAD (<数式>)	ファンクション	<数式>で指定したタブレット，マウス，ライトペン，トラックボールの状態について調べ，その結果を返す．
* PAINT {(X,Y) ｜ STEP (X,Y)} [,<色> [,<境界色>]]	ステートメント	指定された<境界色>で囲まれた領域を，<色>で塗りつぶす．
PDL (<パドル番号>)	ファンクション	指定された番号のパドルの状態を返す．
PEEK (<アドレス>)	ファンクション	<アドレス>で指定されたメモリ1バイトの内容を返す．
PLAY <文字式1> [,<文字式2> [,<文字式3>]]	ステートメント	<文字式（ミュージックマクロ）>にしたがって音楽を演奏する．
PLAY (<ボイスチャンネル>)	ファンクション	音楽が演奏中かどうか調べ，その結果を返す（演奏中なら−1）．
POINT (X,Y)	ファンクション	(X,Y) で指定された座標のドットの色を返す．
POKE <アドレス>,<データ>	ステートメント	<アドレス>で指定されたメモリに<データ>1バイトを書き込む．
POS (<数式>)	システム変数	テキスト画面でのカーソルの水平位置を持つ．代入はできない．
* PRESET {(X,Y) ｜ STEP (X,Y)} [,<色> [,<ロジカルオペレーション>]]	ステートメント	グラフィック画面の (X,Y) で指定された座標のドットを消す．
PRINT [<式> [{;｜,}<式>・・・]]	ステートメント	画面に文字や数字を表示する．
PRINT USING<書式>;<式> [{;｜,}<式>・・・]	ステートメント	<書式>に基づいて画面に文字や数字を表示する．
PRINT #<ファイル番号>, [<式> [{;｜,}<式>・・・]]	ステートメント	<ファイル番号>で指定されたファイルに文字や数字を書き込む．

PRINT　#<ファイル番号>, USING　<書式>；<式>［{；¦，}<式>・・・］

ステートメント　　<書式>に基づいて<ファイル番号>で指定されたファイルに文字や数字を書き込む.

＊ PSET　{(X, Y)¦STEP (X, Y)}［, <色>［, <ロジカルオペレーション>］］

ステートメント　　グラフィック画面の（X, Y）で指定された座標にドットを描く.

＊ PUT KANJI［(X, Y)］, <JIS 漢字コード>［, <色>［, <ロジカルオペレーション>［, <モード>］］］

ステートメント　　画面に漢字を表示する（漢字 ROM が必要）.

＊ PUT SPRITE　<スプライト面番号>［, {(X, Y)¦STEP (X, Y)}［, <色>［, <スプライトパターン番号>］］］

ステートメント　　スプライトパターンを表示する.

R

READ　<変数名>［, <変数名>・・・］

ステートメント　　DATA 文のデータを読み込み，変数に代入する.

REM

ステートメント　　プログラムにコメントを入れる.

RENUM　［<新行番号>［, <旧行番号>［, <増分>］］］

コマンド　　行番号を付け直す.

RESTORE　［<行番号>］

ステートメント　　READ 命令で読み始める DATA の行番号を指定する.

RESUME　{［0］¦NEXT¦<行番号>}

ステートメント　　エラー回復処理を終了し，プログラムの実行を再開する.

RETURN　［<行番号>］

ステートメント　　サブルーチンから戻る.

RIGHT$(<文字式>, <数式>)

ファンクション　　<文字式>の右側から<数式>個分の文字列を取り出す.

RND［(<数式>)］

ファンクション　　0 以上 1 未満の乱数を返す.

RUN　［<行番号>］

コマンド　　<行番号>からプログラムを実行する.

S

SAVE　"<ファイル名>"
コマンド　　プログラムをアスキーセーブする．

＊ SCREEN　<画面モード> [, <スプライトサイズ> [, <キークリックスイッチ> [, <カセットボーレート> [, <プリンタオプション> [, <インターレースモード>]]]]]
ステートメント　　画面モード，その他を設定する．

＊ SET ADJUST（<X 座標オフセット>, <Y 座標オフセット>）
ステートメント　　画面の表示位置を変える．範囲はそれぞれ－7～8．

＊ SET BEEP　<音色>, <音量>
ステートメント　　BEEP 音を選択する．範囲はそれぞれ 1～4．

＊ SET DATE　<文字式> [, A]
ステートメント　　日付を設定する．A はアラームの指定．

＊ SET PAGE　<ディスプレイページ>, <アクティブページ>
ステートメント　　表示するページと，データの読み書きを行うページを指定する．

＊ SET PASSWORD　<文字式>
ステートメント　　パスワードを設定する．

＊ SET PROMPT　<文字式>
ステートメント　　プロンプトを設定する（6 文字まで）．

＊ SET SCREEN
ステートメント　　現在設定されている SCREEN 命令のパラメータを保存する．

＊ SET TIME　<文字式> [, A]
ステートメント　　時刻を設定する．A はアラームの指定．

＊ SET TITLE　<文字式> [, <タイトル色>]
ステートメント　　リセット時に表示されるタイトル画面のタイトル文字列と色を設定する．

＊ SET VIDEO　[<モード> [, <Ym> [, <CB> [, <同期> [, <音声> [, <ビデオ入力> [, <AV コントロール>]]]]]]]
ステートメント　　スーパーインポーズその他のモードを設定する（オプション機能）．

SGN（<数式>）
ファンクション　　符号を調べ，その結果を返す（正＝1，零＝0，負＝－1）．

SIN (<数式>)
ファンクション　　<数式>の正弦（サイン）を返す．角度の単位はラジアン．

SOUND　<レジスタ番号>, <データ>
ステートメント　　PSGのレジスタに値を書き込む．

SPACE$ (<数式>)
ファンクション　　<数式>の長さ分の空白を持った文字列を返す．

SPC (<数式>)
ファンクション　　PRINT系の命令中で<数式>分の空白を出力する．

SPRITE　{ON | OFF | STOP}
ステートメント　　スプライトの重なりによる割り込みを許可，禁止，保留する．

SPRITE$ (<スプライトパターン番号>)
システム変数　　スプライトパターンを持つ．

SQR (<数式>)
ファンクション　　<数式>の平方根を返す．

STICK (<ジョイスティック番号>)
ファンクション　　ジョイスティックの押されている方向を調べ，その結果を返す．

STOP
ステートメント　　プログラムの実行を停止する．

STOP　{ON | OFF | STOP}
ステートメント　　CTRL+STOPによる割り込みを許可，禁止，保留する．

STRIG (<ジョイスティック番号>)
ファンクション　　トリガボタンの状態を調べ，その結果を返す．

STRIG (<ジョイスティック番号>) {ON | OFF | STOP}
ステートメント　　トリガボタンによる割り込みを許可，禁止，保留する．

STR$ (<数式>)
ファンクション　　<数式>の値を10進表記の文字列に変換し，その結果を返す．

STRING$ (<数式1>, {<文字式> | <数式2>})
ファンクション　　<文字式>の先頭文字または<数式2>のコードを持つ文字を<数式1>分の長さの文字列にして返す．

SWAP　<変数名>, <変数名>	
ステートメント	2つの変数の値を交換する.

T

TAB (<数式>)	
ファンクション	PRINT系の命令中で, 指定された桁まで空白を出力する.
TAN (<数式>)	
ファンクション	<数式>の正接（タンジェント）を返す. 角度の単位はラジアン.
TIME	
システム変数	インターバルタイマの値を持つ.
TRON	
コマンド	実行しているプログラムの行番号を画面に表示し続ける.
TROFF	
コマンド	TRONをキャンセルし, 行番号の表示をやめる.

U

USR [<番号>] (<引数>)	
ファンクション	マシン語ルーチンを呼び出す.

V

VAL (<文字式>)	
ファンクション	<文字式>を数値に変換し, その結果を返す.
VARPTR (<変数名>)	
ファンクション	変数の格納されているアドレスを返す.
VARPTR (#<ファイル番号>)	
ファンクション	ファイル・コントロールブロックの開始アドレスを返す.
*** VDP (<レジスタ番号>)**	
システム変数	VDPのレジスタに対し, データの読み書きをする.

＊ VPEEK (<アドレス>) ファンクション　　VRAMの<アドレス>からデータを読み出す．	
＊ VPOKE　<アドレス>, <データ> ステートメント　　VRAMの<アドレス>にデータを書き込む．	

W

WAIT　<ポート番号>, <数式1> [, <数式2>] ステートメント　　入力ポートのデータが指定された値になるまで，実行を中断する．	
＊ WIDTH　<桁数> ステートメント　　画面の1行の桁数を指定する．	

1.2　MSX DISK-BASICの命令

B

＊ BLOAD　"<ファイル名>" [{[, R] ¦ [, S]} [, <オフセット>]] コマンド　　マシン語プログラムや画面データを，ファイルからロードする．	
＊ BSAVE　"<ファイル名>", <開始アドレス>, <終了アドレス> [, {<実行開始アドレス> ¦ S}] コマンド　　マシン語プログラムや画面データを，ファイルにセーブする．	

C

CLOSE　[[#]<ファイル番号> [, [#]<ファイル番号>・・・]] ステートメント　　<ファイル番号>に対応するファイルをクローズする．	
CALL FORMAT コマンド　　フロッピーディスクをフォーマットする．	
CALL SYSTEM コマンド　　MSX-DOSに戻る．	

COPY　“<ファイル名1>” [TO “<ファイル名2>”]

コマンド　　<ファイル名2>で指定されたファイルに<ファイル名1>の内容をコピーする．

CVD（<8バイト文字列>）

ファンクション　　文字列を倍精度実数値に変換し，その結果を返す．

CVI（<2バイト文字列>）

ファンクション　　文字列を整数値に変換し，その結果を返す．

CVS（<4バイト文字列>）

ファンクション　　文字列を単精度実数値に変換し，その結果を返す．

D

DSKF（<ドライブ番号>）

ファンクション　　ディスクの残りの容量をクラスタ単位で返す．

E

EOF（<ファイル番号>）

ファンクション　　ファイルの終わりに達したか否かを調べ，終わりなら−1を返す．

F

FIELD　[#]<ファイル番号>，<フィールド幅>　AS　<文字変数名> [，<フィールド幅>　AS　<文字変数名>・・・]

ステートメント　　ランダム入出力用バッファに，文字変数を割り当てる．

FILES　[“<ファイル名>”]

コマンド　　<ファイル名>に一致したファイルの名前を画面に表示する．

G

GET　[#] <ファイル番号> [，<レコード番号>]

ステートメント　　ランダムファイルから，ランダム入出力用バッファに1レコード読み込む．

I

INPUT #<ファイル番号>, <変数名> [, <変数名> ・・・]
ステートメント　　ファイルからデータを読み込む.

INPUT$(<文字数> [, [#]<ファイル番号>])
ファンクション　　ファイルから，指定された長さの文字列を得る.

K

KILL "<ファイル名>"
コマンド　　<ファイル名>で指定したファイルを削除する.

L

LFILES ["<ファイル名>"]
コマンド　　<ファイル名>に一致したファイルの名前をプリンタに出力する.

LINE INPUT #<ファイル番号>, <文字変数名>
ステートメント　　1行単位のデータをファイルから文字変数へ読み込む.

LOAD "<ファイル名>" [, R]
コマンド　　プログラムをメモリ上にロードする.

LOC (<ファイル番号>)
ファンクション　　ファイル中で最後にアクセスが行われた位置をレコード番号で返す.

LOF (<ファイル番号>)
ファンクション　　指定されたファイルの大きさをバイト単位で返す.

LSET <文字変数名>=<文字式>
ステートメント　　ランダム入出力用バッファに，データを左詰めにして格納する.

M

MAXFILES=<ファイル数>
ステートメント　　オープン可能な最大のファイル数を宣言する.

MERGE "<ファイル名>"
コマンド　　メモリ上のプログラムとアスキーセーブされているプログラムを混合する.

MKD$(<倍精度実数値>)
ファンクション　　倍精度実数を内部表現に対応したキャラクタコードに変換する.

MKI$(<整数値>)
ファンクション　　整数を内部表現に対応したキャラクタコードに変換する.

MKS$(<単精度実数値>)
ファンクション　　単精度実数を内部表現に対応したキャラクタコードに変換する.

N

NAME　"<ファイル名1>"　AS　"<ファイル名2>"
コマンド　　ファイルの名前を変更する.

O

OPEN　"<ファイル名>"[FOR　<モード>]　AS　#<ファイル番号> [LEN=<レコード長>]
ステートメント　　ファイルをオープンする.

P

PRINT　#<ファイル番号>, [<式> [{;|,}<式>・・・]]
ステートメント　　シーケンシャルファイルにデータを出力する.

PRINT　#<ファイル番号>, USING　<書式>;<式> [{;|,}<式>・・・]
ステートメント　　書式に基づいてシーケンシャルファイルにデータを出力する.

PUT　[#]<ファイル番号> [,<レコード番号>]
ステートメント　　ランダム入出力用バッファ中のデータをランダムファイルに出力する.

R

RSET　<文字変数名>=<文字式>
ステートメント　　ランダム入出力用バッファに，データを右詰めにして格納する.

RUN　"<ファイル名>"[, R]
コマンド　　ディスクからプログラムをロードし，実行する.

S

SAVE　“<ファイル名>”[, A]

コマンド　　　　プログラムをセーブする．Aを指定するとアスキーセーブされる．

V

VARPTR (#<ファイル番号>)

ファンクション　　　　ファイル・コントロールブロックの開始アドレスを返す．

2章 MSX BASIC ver 2.0 の変更点

MSX BASIC ver 2.0 は，MSX BASIC ver 1.0 に比べ大幅に機能が追加，変更されています．それらは，VDP（Video Display Processor）のバージョンアップに伴うものと，RAM ディスク，時計，メモリスイッチなどの諸機能がハードウェア的に追加されたことによるものですが，特に VDP の変更は画面関係のステートメントのほとんどになんらかの影響を与えています．

本章ではそれらのステートメントをピックアップし，追加，変更のあった箇所を示します．なお，文中の "MSX 1" とは MSX BASIC ver 1.0 を，"MSX 2" とは MSX BASIC ver 2.0 を指します．

2.1 スクリーンモードに関する追加，変更

● SCREEN [<画面モード>[,<スプライトサイズ>[,<キークリックスイッチ>[,<カセットボーレート>[,<プリンタオプション>[,<インターレースモード>]]]]]]

変更があったのは<画面モード>と<インターレースモード>です．

<画面モード>は，0～8 まで指定することができるようになりました．このうち 0～3 までは MSX1 と同じで，残りの 4～8 が新設されたモードです．ひとつの画面モードを指す場合，BASIC では "SCREEN モード" という呼び方をしますが，これは VDP の内部で使用している "画面モード" とは若干の違いがあります．これらの対応と，その意味を**表 2.1** に示します．なお，画面モード 2 と 4 の違いはスプライトの表示機能だけです．

<インターレースモード>を指定することで，VDP のインターレース機能の設定ができます（**表 2.2**)．2 画面交互表示モードの場合は，"SET PAGE"で指定するディスプレイページが奇数ページでなければなりません．また，その時はディスプレイページとそれよりひとつ番号の小さいページが交互に表示されることになります．

BASICのモード	VDPのモード	意　　味		
		ドットまたは文字数	同時表示色	スクリーン形式
SCREEN0：WIDTH40	TEXT1 MODE	40×24文字	512色中　2色	テキスト
SCREEN0：WIDTH80	TEXT2 MODE	80×24文字	512色中　2色	テキスト
SCREEN1	GRAPHIC1 MODE	32×24文字	512色中　16色	テキスト
SCREEN2	GRAPHIC2 MODE	256×192ドット	512色中　16色	高解像度グラフィック
SCREEN3	MULTI COLOR MODE	64×48ドット	512色中　16色	低解像度グラフィック
SCREEN4	GRAPHIC3 MODE	256×192ドット	512色中　16色	高解像度グラフィック
SCREEN5	GRAPHIC4 MODE	256×212ドット	512色中　16色	ビットマップグラフィック
SCREEN6	GRAPHIC5 MODE	512×212ドット	512色中　4色	ビットマップグラフィック
SCREEN7	GRAPHIC6 MODE	512×212ドット	512色中　16色	ビットマップグラフィック
SCREEN8	GRAPHIC7 MODE	256×212ドット	256色	ビットマップグラフィック

表2.1　BASIC画面(SCREEN)モードと，VDPの画面モードの対応

インターレースモード	表示機能
0	ノン・インターレース通常表示(省略値)
1	インターレース表示
2	ノン・インターレース，Even／Odd 2画面交互表示
3	インターレース，Even／Odd 2画面交互表示

表2.2　インターレースモードによる表示機能の違い

● SET PAGE ＜ディスプレイページ＞，＜アクティブページ＞

これは新たに追加されたステートメントで，画面に表示するページと，データを読み書きするページの設定を行うためのものです．これは画面モードが5～8の場合に有効で，指定できる値はVRAMの容量と画面モードによって異なります（**表2.3**)．

VRAM上でのページの割り振られ方は，APPENDIXのVRAMマップを参照してください．

画面モード	VRAM64K	VRAM128K
SCREEN5	0～1	0～3
SCREEN6	0～1	0～3
SCREEN7	使用不可	0～1
SCREEN8	使用不可	0～1

表2.3　画面モードとVRAM容量によるページ指定値の範囲

● WIDTH ＜桁数＞

画面モードが0の場合，＜桁数＞は80まで指定することが可能になりました．

2.2　カラーの指定に関する追加，変更

● **COLOR　[<前景色>[,<背景色>[,<周辺色>]]]**

MSX 2では，カラーパレット機能が使えるようになったために，画面モードによって色を指定する数値の範囲や意味が異なってきます（**表2.4**）．なお，<背景色>はテキスト画面以外ではCLS文実行時に変わります．

画面モード6における<周辺色>は特殊な意味を持ちます．**図2.2**は画面モード6の時の<周辺色>のビットごとの意味ですがこのモードではフラグ（ビット4）の操作により，X座標の奇数番の縦ラインと偶数番の縦ラインの色を別々に指定することができます．

フラグが0（周辺色値が0～15）の時は別々の色を指定することはできず，奇数番縦ラインの色で周辺色が設定されます．フラグが1（周辺色値が16～31）の時は偶数番縦ラインと奇数番縦ラインの色で周辺色が設定され，その2つの色が異なっていた場合は画面は縦じま模様になります．

7	6	5	4	3	2	1	0
未使用			フラグ	偶数ラインの色		奇数ラインの色	

図2.2　画面モード6のときの周辺色のビットごとの意味

● **COLOR=(<パレット番号>,<赤の輝度>,<緑の輝度>,<青の輝度>)**

指定したパレットに色を設定します．<パレット番号>の指定は，**表2.4**を参照してください．ただし，画面モード8の場合はパレット機能がないので何も起こりません（エラーにもならない）．なお，パレット番号の0番は，通常は透明色（周辺が透けて見える）に固定されていますが，VDPのレジスタを変更することにより，他のパレットと同様に扱えるようになります．

```
VDP (9)=VDP (9) OR &H20 ………… 同様に扱う時
VDP (9)=VDP (9) AND &HDF ……… 透明色に固定したい時
```

画面モード	色指定	番号の範囲
SCREEN 0	パレット番号	0～15
SCREEN 1	パレット番号	0～15
SCREEN 2	パレット番号	0～15
SCREEN 3	パレット番号	0～15
SCREEN 4	パレット番号	0～15
SCREEN 5	パレット番号	0～15
SCREEN 6	パレット番号	0～3
SCREEN 7	パレット番号	0～15
SCREEN 8	色番号	0～255

表2.4　画面モードごとの色指定

各色の輝度は0〜7までの8段階指定することができ，その組み合わせにより8(赤)×8(緑)×8(青)=512色を表現することが可能です．

● COLOR=RESTORE

カラーパレット保存テーブル（APPENDIX VRAM マップ参照）の内容のとおりに，カラーパレット・レジスタを再設定します．たとえば通常とは異ったカラーパレットの設定で描いた画像データをBSAVEした場合，それをBLOADしただけではカラーパレットは変わらないので，はじめの画像は再現できません．そのような場合は，カラーパレット保存テーブルを含めて画像をBSAVEしておきます．そのデータをBLOADした後に，COLOR=RESTORE命令でパレットを再設定することにより，はじめの画像と同じ色を得ることができます．

● COLOR [=NEW]

カラーパレットを，電源投入時と同じ状態に初期化します(**表2.5**)．プログラムの最初と最後に入れておくとよいでしょう．

パレット番号	色	赤の輝度	青の輝度	緑の輝度
0	透明	0	0	0
1	黒	0	0	0
2	緑	1	1	6
3	明るい緑	3	3	7
4	暗い青	1	7	1
5	明るい青	2	7	3
6	暗い赤	5	1	1
7	シアン	2	7	6
8	赤	7	1	1
9	明るい赤	7	3	3
10	暗い黄	6	1	6
11	明るい黄	6	3	6
12	暗い緑	1	1	4
13	マゼンタ	6	5	2
14	灰	5	5	5
15	白	7	7	7

表2.5　カラーパレットの初期化色とパレット設定値

2.3　文字の表示に関する追加，変更

● LOCATE [<X 座標> [, <Y 座標> [, <カーソルスイッチ>]]]

テキスト画面上で文字を表示する位置を指定する命令です．

画面モード 0 に80桁表示機能が追加されたため，この機能の使用時には，X座標は最大 79 まで指定可能となりました．

2.4　グラフィックスの表示に関する追加，変更

● LINE [{(X1, Y1) ¦ STEP (X1, Y1)}]－{(X2, Y2) ¦ STEP (X2, Y2)} [, <色>[, {B ¦ BF} [, <ロジカルオペレーション>]]]

● PSET {(X, Y) ¦ STEP (X, Y)} [, <色>[, <ロジカルオペレーション>]]

● PRESET {(X, Y) ¦ STEP (X, Y)} [, <色>[, <ロジカルオペレーション>]]

これらは，画面モードに応じて指定できる座標の範囲が異なります（**表 2.6**）．

画面モード	X座標	Y座標
SCREEN 2	0～255	0～191
SCREEN 3	0～255	0～191
SCREEN 4	0～255	0～191
SCREEN 5	0～255	0～211
SCREEN 6	0～511	0～211
SCREEN 7	0～511	0～211
SCREEN 8	0～255	0～211

表2.6　画面モードによる座標の指定範囲

新たにロジカルオペレーションの機能が追加されました．<ロジカルオペレーション>を指定すると，指定した<色>ともとの色の間で論理演算が行われ，実際にはその結果の色で描かれます．ロジカルオペレーションの種類は**表 2.7** のとおりです．なお<色>の指定は，画面モード 8 を除いてパレット番号で行います．

ロジカルオペレーション	機　　能
PSET（省略値），TPSET	"指定色" で描く
PRESET　　　，TPRESET	"NOT（指定色）" で描く
XOR　　　　，TXOR	"（背景色）XOR（指定色）" で描く
OR　　　　，TOR	"（背景色）OR（指定色）" で描く
AND　　　　，TAND	"（背景色）AND（指定色）" で描く

注）〈色〉を（指定色）とする
描かれる所のもとの色を（背景色）とする
頭に "T" がついたロジカルオペレーションを指定すると〈色〉が透明色（COLOR 0）の場合，何も行われない

表2.7　ロジカルオペレーション

● **CIRCLE {(X, Y)¦STEP (X, Y)}, ＜半径＞[, ＜色＞[, ＜開始角度＞[, ＜終了角度＞ [, ＜比率＞]]]]**

画面モードに応じて，指定できる座標の範囲が異なってきます（2.6参照)．＜色＞の指定は，画面モード8を除いてパレット番号で行います．

● **PAINT {(X, Y)¦STEP (X, Y)} [, ＜色＞ [, ＜境界色＞]]**

画面モードに応じて，指定できる座標の範囲が異なってきます（表2.6参照)．＜色＞の指定は，画面モード8を除いてパレット番号で行います．画面モード2と4では＜境界色＞の指定は無効となります．

● **PUT KANJI [(X, Y)], ＜漢字コード＞[, ＜色＞[, ＜ロジカルオペレーション＞ [, ＜モード＞]]]**

漢字を表示します．＜漢字コード＞は，JIS漢字コードで指定してください．＜漢字コード＞に指定できる値の有効範囲は&H2121～&H4F53で，厳密には上位バイトが&H21～&H27, &H30～&H4Fで，下位バイトが&H21～&H7Eの値となります．＜ロジカルオペレーション＞については表2.7を参照してください．＜モード＞は，**表2.8**のような意味を持ちます．なお，この命令は画面モード5～8で有効です．

モード	機　　能
0	16×16ドットで普通に表示する（省略値）
1	偶数番目の線だけを，縦8ドットで表示する
2	奇数番目の線だけを，縦8ドットで表示する

表2.8　PUT KANJIの〈モード〉

2.5　VDPのアクセスに関する追加，変更

● **BASE（＜式＞）**

VRAMに割り振られている，各テーブルの先頭アドレスを返します．＜式＞と各画面モードのテーブルは，**表2.9**のように対応しています．

テーブル先頭アドレスの読み出しはすべての＜式＞に対して可能ですが，書き込みは＜式＞が0～19（画面モード0～3まで）の時しかできません．

画面モードの2のテーブルアドレスを変更すると，同時に画面モード4のテーブルも変わるので注意してください．

画面モード5～8の場合の返されるアドレスは，アクティブページの先頭アドレスからのオフセット値です．

式	画面モード	テーブル
0	0	パターンネーム・テーブル
1	0	―――
2	0	パターンジェネレータ・テーブル
3	0	―――
4	0	―――
5	1	パターンネーム・テーブル
6	1	カラーテーブル
7	1	パターンジェネレータ・テーブル
8	1	スプライトアトリビュート・テーブル
9	1	スプライトジェネレータ・テーブル
10	2	パターンネーム・テーブル
11	2	カラーテーブル
12	2	パターンジェネレータ・テーブル
⋮	⋮	⋮
43	8	スプライトアトリビュート・テーブル
44	8	スプライトジェネレータ・テーブル

表2.9　BASEの設定値とVRAMテーブルの対応

● VDP（<n>）

VDPレジスタの値の読み出しと書き込みを行います．<n>の部分は，実際のVDPのレジスタの番号とは若干のズレがあります．<n>と実際のVDPのレジスタの対応を**表2.10**に示します．

n	VDPレジスタ番号	アクセスモード
0～7	0～7（MSX1と同じ）	リード／ライト
8	ステータスレジスタ0	リードのみ
9～24	8～23	リード／ライト
33～47	32～46	ライトのみ
−1～−9	ステータスレジスタ1～9	リードのみ

表2.10　VDPレジスタとの対応

● VPEEK（<アドレス>）

● VPOKE　<アドレス>，<データ>

画面モードが5～8の時は，アクティブページの先頭アドレスからのオフセット値を<アドレス>に設定します．<アドレス>に入る値の有効範囲は0～65535で，<データ>に入る値の有効範囲は0～255です．

2.6　画像データの保存に関する追加，変更

- BSAVE ＜ファイル名＞，＜開始アドレス＞，＜終了アドレス＞，S
- BLOAD ＜ファイル名＞，S

DISK BASICのステートメントで，VRAMの内容をディスクファイルにセーブ／ロードする時に使用します．どの画面モードでも有効ですが，画面モード5～8の時はアクティブページが対象になります．なお，カセットテープに対しては使用できません．＜アドレス＞に入る値の有効範囲は −32768～−2，0～65534（&HFFFE）です．

- COPY (X1, Y1)−(X2, Y2) [,＜ソースページ＞] TO (X3, Y3) [,＜デスティネーションページ＞[,＜ロジカルオペレーション＞]]
- COPY (X1, Y1)−(X2, Y2) [,＜ソースページ＞] TO {＜配列変数名＞ ¦ ＜ファイル名＞}
- COPY {＜配列変数名＞ ¦ ＜ファイル名＞} [,＜向き＞] TO (X3, Y3) [,＜デスティネーションページ＞[,＜ロジカルオペレーション＞]]
- COPY ＜ファイル名＞ TO ＜配列変数名＞
- COPY ＜配列変数名＞ TO ＜ファイル名＞

COPY命令は画像データ転送用のステートメントで，画面モード5～8の時有効です．対象となるのはVRAM，配列変数，ディスクファイルのいずれかで，それらの間を自由にデータ転送することができます．

(X1, Y1)−(X2, Y2) は，この2つの座標を対角線とする四角い領域を転送することを示します．＜ソースページ＞は転送元の，＜ディスティネーションページ＞は転送先のページを表し，省略するとアクティブページが指定されます．＜向き＞は，画像データを画面に書き込む際の向きを表し，0～3の範囲で指定します（図2.3）．

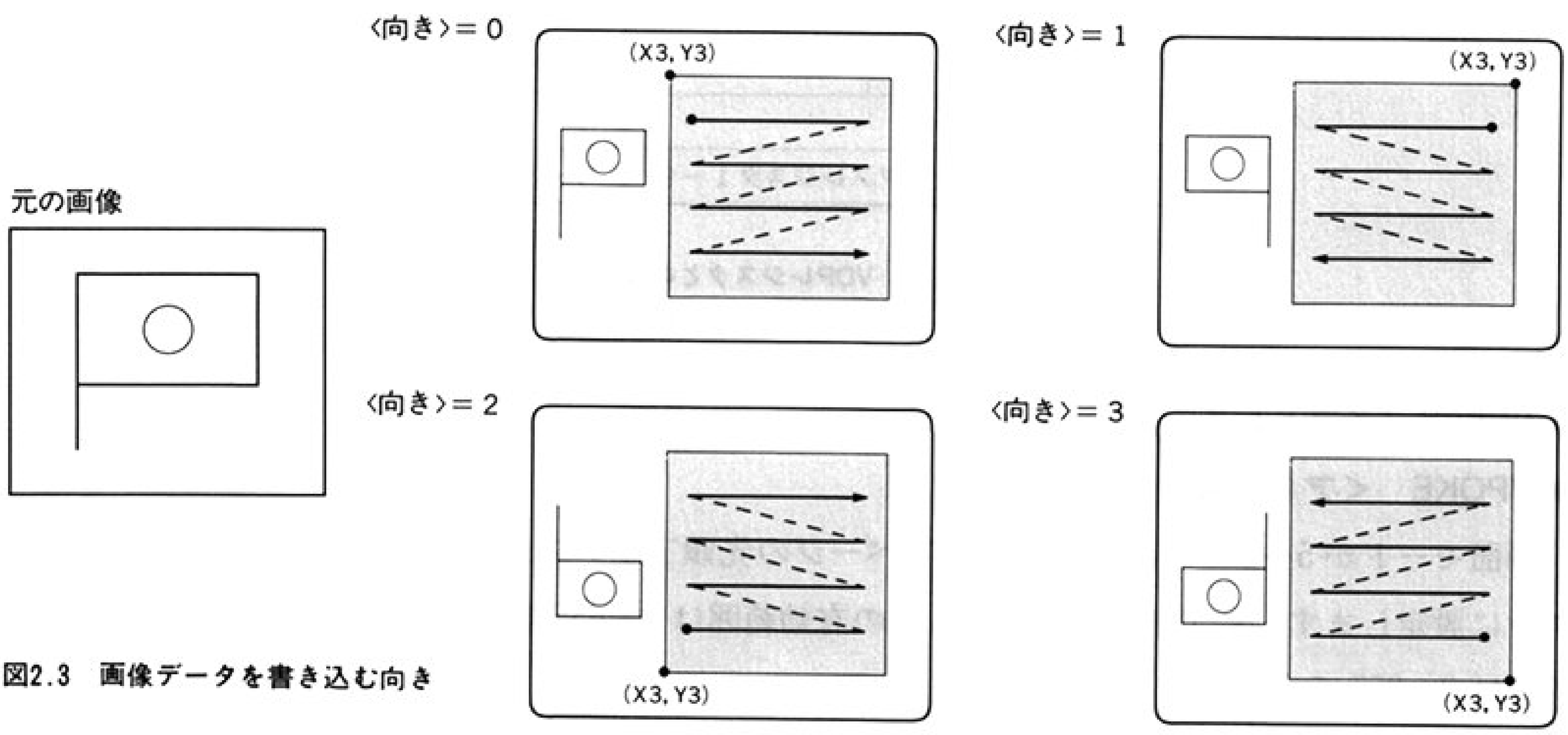


図2.3　画像データを書き込む向き

＜配列変数＞は，整数型，単精度実数型，倍精度実数型のいずれかの配列変数で，あらかじめ画像データを取り込める大きさの領域を確保しておかなければなりません．大きさは，**式1**で算出することができます．＜ピクセルサイズ＞は，画面の1ドットを表すために使うビット数で，画面モード5と7の場合は4，画面モード6の場合は2，画面モード8の場合は8が入ります．また，画像データは図2.4のフォーマットで記憶されます．

式1

INT ((<ピクセルサイズ>*(ABS (X2−X1)+1) *(ABS (Y2−Y1)+1)+7)/8)+4　バイト

横幅の下位バイト	0
横幅の上位バイト	1
縦幅の下位バイト	2
縦幅の上位バイト	3
画像データ※	4 ⋮ nバイト

※データの長さがバイトで割り切れなければ余ったビットは0となる

図2.4　画像データのフォーマット

＜ロジカルオペレーション＞は，転送する先にあるデータと転送するデータの論理演算を指定します．指定するパラメータは表2.7を参照してください．なお頭に“T”をつけた場合は，転送元の透明色の部分は転送されません．

2.7　スプライトに関する追加，変更

MSX 2の画面モード4～8で使用するスプライトは，スプライトモード2と呼ばれ，MSX 1のものと比べかなりの機能アップがされています．たとえばMSX 1では1枚のスプライトで扱うことができるのは1色だけでしたが，MSX2のこのモードでは横の各ラインに個別の色が指定でき，それにより多色のキャラクタを1枚のスプライトで実現できるようになりました．さらに，ドットごとに色をつけたい場合などは，2枚以上のスプライトを組み合わせて1枚のスプライトのように使用することもできるので，それらを併用するとよいでしょう．また，MSX 1では5個以上のスプライトが横ライン上に並んだ場合5個目以降のスプライトは表示されませんでしたが，MSX 2では8個まで表示可能になり，より自由度の高いものとなっています．

スプライトに指定できる色は画面モード8を除いて表2.4(COLOR文)と同じですが，画面モード8のスプライトはパレットが使えないため指定は色番号となり，使用できる色は**表2.11**に示す16色のみとなります．

0：黒	1：暗い青	2：暗い赤	3：暗い紫	4：暗い緑	5：暗い水色	6：暗い黄色	7：灰色
8：肌色	9：青	10：赤	11：紫	12：緑	13：水色	14：黄	15：白

表2.11　画面モード8におけるスプライト指定色

● **PUT SPRITE　<スプライト面番号> [, [STEP] (X, Y) [, <色>[, <スプライトパターン番号>]]]**

画面モード1～3では指定したスプライト面の表示を消す場合はY座標を209，指定したスプライト面以降のすべてを消す場合はY座標を208としましたが，画面モード4～8においてはY座標が212ドットに増えたため，指定値がそれぞれ217，216に変わりました．

● **COLOR SPRITE$(<スプライト面番号>)=<文字式>**

スプライトの横ラインごとの色を指定します（図2.5）．

<文字式>に指定できるのは1～16文字で，その文字のアスキーコードの0～3ビットで色を指定し，4～7ビットでスプライトの各機能を指定します（表2.12）．なお，これらの指定は画面モード4～8の場合にのみ有効です．

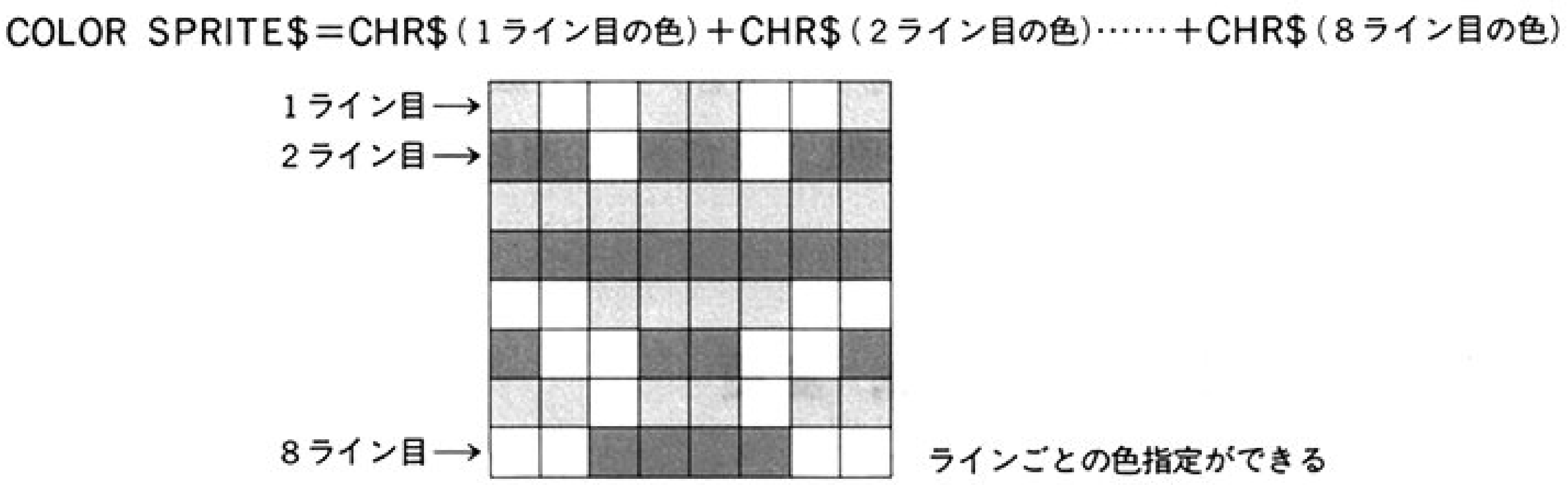


図2.5　スプライトと〈文字式〉の関係

b7	1ならば，スプライトを32ドット左にずらす
b6	1ならば，連続したスプライト面のスプライトを同時に動かす．このときスプライトの優先順位と衝突は無視され，スプライトが重なった場合それらの色番号をORした色で表示される※
b5	1ならば，スプライトの衝突を無視する
b4	未使用
b0～b3	パレット番号

※ たとえばスプライト面1のビット6を'0'，スプライト面2のビット6を'1'としておくと，スプライト面1を動かすだけでスプライト面2が同じ位置に重なって表示される．

表2.12　文字式のビットごとの意味

● COLOR SPRITE（<スプライト面番号>）=<数式>

指定された面のスプライト全体を＜数式＞の色にします．さきほどのCOLOR SPRITE$は色指定を＜文字式＞で行いますが，これは＜数式＞で行うところが違います．色指定の形式は表2.12と同じですが，b7の指定はできません．なお，これらは画面モード4～8の時に有効です．

2.8　オプション機能に関する追加

● SET VIDEO<モード>[,<Ym> [,<CB>[,<同期>[,<音声>[,<ビデオ入力> [,<AVコントロール>]]]]]]

このステートメントは，オプションであるスーパーインポーザやディジタイザに対するステートメントであり，それらの機能を装備している機種に対してのみ使用できます．

＜モード＞はスーパーインポーズのモード設定をするもので，**表2.13**に示す値が指定できます．

＜Ym＞が1の時，テレビの輝度が半分になります．

＜CB＞が1の時，VDPのカラーバスが入力状態になり，0の時は，出力状態になります．

＜同期＞が1の時「外部同期」が，0の時「内部同期」が選択されます．

＜音声＞は，外部信号を混合して出力するかどうかの指定で，**表2.14**の値が入ります．

＜ビデオ入力＞は，外部映像信号の入力を切り換え，0ならRGBマルチコネクタが，1なら外部映像信号入力コネクタが選択されます．

＜AVコントロール＞は，RGBマルチコネクタのAVコントロール端子の出力を指定するためのもので，0ならOFFに，1ならONになります．

モード	S1	S0	TP	表示画面
0	0	0	0	コンピュータ
1	0	1	1	コンピュータ
2	0	1	0	スーパーインポーズ
3	1	0	0	テレビ

注）モード0の場合は，外部同期をかけることはできません．その他のモードでは，VDPのコンポジット出力は使えません．S1，S0，TPはVDPのレジスタ内のフラグの名称です．

表2.13　SET VIDEOの〈モード〉の入力値

音声	機能
0	外部音声信号を混合しない
1	右チャンネルの外部音声信号を混合する
2	左チャンネルの外部音声信号を混合する
3	両チャンネルの外部音声信号を混合する

表2.14　SET VIDEOの〈音声〉の入力値

● COPY SCREEN [<モード>]

ディジタイズなどを行い，カラーバスのデータをVRAMに書き込む時に使用します．これは画面モード5～8のみ有効です．

＜モード＞が0の時は，1フィールドの信号をディジタイズしてディスプレイページに書き込み，＜モード＞が1の時は，連続する2フィールド（つまり1フレーム）の信号を（ディスプレイページ－1）番のページとディスプレイページに書き込みます．ですからモードが1の場合は，ディスプレイページが奇数番のページになっている必要があります．なお，省略した場合は，モードが0となります．

2.9　時計機能に関する追加

● GET DATE <文字変数名>[, A]

時計から日付を読み，文字変数に代入するための命令です．読み込まれる日付の形式は次のようになっています．

YY/MM/DD　　　　（YY＝西暦の下2ケタ，MM＝月，DD＝日）

例）85/03/23（1985年3月23日）

なお，オプションのAを指定するとアラームの日付が読み込まれます．

● SET DATE <文字式>[, A]

時計に日付を設定します．パラメータとオプションの形式は“GET DATE”と同じです．

例）　SET DATE　"85/03/23"

● GET TIME <文字変数名> [, A]

時計から時刻を読み，文字変数に代入するためのステートメントです．読み込まれる時刻の形式は次のようになっています．

HH : MM : SS　　　　（HH＝時，MM＝分，SS＝秒）

例）22 : 15 : 00（22時15分0秒）

なおオプションのAを指定するとアラームの時刻が読み込まれます．

● SET TIME <文字式> [, A]

時計に時刻を設定します．パラメータとオプションの形式は“GET TIME”と同じです．

例）　SET TIME　"22 : 15 : 00"

※アラームについて

アラーム機能はオプションなので，指定した時刻に何が起こるかは機種によって異なります(普通はなにも起きない).

"SET DATE"と"SET TIME"の両方のアラーム設定を行う場合は，かならず"SET TIME"から行います("SET TIME"を実行すると，"SET DATE"で設定したアラームの日付が，消えてしまう).

なお，アラームが設定できるのは，分単位です(秒を設定しても無視される).

2.10　メモリスイッチ機能に関する追加

"SET"命令により，CLOCK-IC内のバッテリバックアップRAMに，以下に挙げるいろいろな設定を記憶させておくことができます．それによってシステム起動時(電源投入時やリセット時)にそれらにしたがった設定が自動的に行われます．なお，"SET TITLE と"SET PROMPT"，"SET PASSWORD"は同一RAMを使用するため，最後に実行された命令による設定のみが有効になります．

● **SET ADJUST(<X座標オフセット>，<Y座標オフセット>)**

画面の表示位置を設定します．座標オフセットの設定値は，－7～8です．

● **SET BEEP　<音色>，<音量>**

BEEP音を設定します．<音色><音量>の設定値は，ともに1～4です．

<音色>と実際の音の対応を，**表2.15**に示します．

音色	音
1	ぴー(MSX1と同じ)
2	ぽぉ
3	ぴぃんぽぉん
4	ぽぴぷっ

表2.15　SET BEEPの〈音色〉の入力値

● **SET TITLE　<タイトル>[，<色>]**

システム起動時の初期画面に表示されるタイトルと，その時の画面の色を指定します．<タイトル>には，6文字以内の文字列を設定し，<色>には**表2.16**に示す数値が入ります．<タイトル>がちょうど6文字の場合に限り，タイトル画面を表示したところでキー入力待ちとなります．

〈色〉	1	2	3	4
画面色	青	緑	赤	オレンジ

表2.16 SET TITLEで指定できる色

● SET PROMPT ＜プロンプト＞

プロンプトを設定します. ＜プロンプト＞には6文字以内の文字列を設定します.

● SET PASSWORD ＜パスワード＞

システムのパスワードを設定します. ＜パスワード＞は255文字以内の文字式で, このステートメントを実行した後システムが起動されるとパスワードの入力要求待ちになります. 正しいパスワードが入力された場合は通常どおり起動し, それ以外は再び入力待ちとなります. なお, グラフィックキーとストップキーの両方を押しながらシステム起動すると, パスワードの要求は行われません (ただし, キーカートリッジでパスワードの設定をした場合はかならずパスワード入力をしなければ起動しない). なお, SET TITLEで空文字を指定することにより, パスワードは解除されます.

● SET SCREEN

現在設定されている“SCREEN”文のパラメータを記録します. システム起動時には, それにしたがって自動的に設定がされます. 記録される項目は以下のとおりです.

テキストモードのスクリーン番号,	キークリックスイッチ,
テキストモードの表示幅,	プリンタオプション,
前景色, 背景色, 周辺色,	カセットボーレート,
ファンクションキースイッチ,	ディスプレイモード

2.11 RAMディスク機能に関する追加

MSX 1では, 0000H～7FFFHまでのRAMはDOSでしか使われませんでしたが, MSX 2ではこの部分を最高32KバイトのRAMディスクとして使用できるようになりました. RAMディスクに対するファイル名の形式は以下のようになっており, ＜ファイル名前部＞には1～8文字, ＜拡張子＞には1～3文字の長さの文字列が入ります (ただし, “:”(コロン), “.”(ピリオド) およびキャラクタコードの00H～1FHのコントロールキャラクタと2バイトのグラフィック記号は使用できない).

MEM:＜ファイル名前部＞[.＜拡張子＞]

RAM ディスクで実行可能な操作と，それに関係するステートメントは以下のとおりです．

1．BASICプログラムのロード／セーブ（かならずアスキーセーブされる）
SAVE，LOAD，RUN，MERGE

2．シーケンシャルファイルのリード／ライト
OPEN，CLOSE
PRINT　#，PRINT USING　#
INPUT　#，LINE INPUT　#，INPUT$
EOF，LOC，LOF

なお，RAM ディスクでは以下の命令をサポートしていません．

1．ランダムファイルのリード／ライト
2．BLOAD，BSAVE
3．COPY

● CALL MEMINI [(<サイズ>)]

RAM ディスクとして使用するメモリを指定し，RAM ディスクの初期化，全ファイルの消去を行います．RAM ディスクを使用する場合は，かならずこのステートメントを実行しておかなければなりません．

<サイズ>には，“RAM ディスクで使用するメモリ容量－1”が入り，これには1023～32767の値の指定ができます（ただしワークエリアとして768バイト分のRAMを使用）．<サイズ>を省略すると，最高値である32768バイトがRAMディスクに割り振られます．また“CALL MEMINI (0)”を実行することにより，RAM ディスク機能が解除されます．

● CALL MFILES

RAM ディスク内のファイル名を表示します．

● CALL MKILL　(“<ファイル名>”)

指定されたファイルを削除します．

● CALL MNAME　(“<旧ファイル名>” AS “<新ファイル名>”)

ファイル名を変更します．

2.12　その他の追加機能

● PAD（<数式>）

タッチパッド（タッチパネル），ライトペン，マウス，トラックボールの状態を返す関数です．

<数式>が0～7の場合は，MSX1と同様タッチパッドの状態を返し，8～11の場合は，ライトペンの状態を返します．なお，“PAD(8)”を実行した時に座標とスイッチの値を読み込むようになっているので，PAD(8)の値が－1であることを確認し，他のデータを読むようにしてください（表2.17）．

式	意　味
8	ライトペンのデータが有効であれば－1，無効ならば0を返す
9	ライトペンのX座標を返す
10	ライトペンのY座標を返す
11	ライトペンのスイッチが押されていれば－1，いなければ0を返す

表2.17　ライトペンの状態を返す<式>

<数式>が12～15の場合はポート1に，16～19の場合はポート2に接続されたマウスまたはトラックボールの状態を返します（表2.18）．マウスとトラックボールの区別は，自動的に行われます．

式	意　味
12, 16	－1を返す，入力要求に使用する
13, 17	X座標を返す
14, 18	Y座標を返す
15, 19	0を返す（未使用）

表2.18　マウスまたはトラックボールの状態を返す<式>

座標データは，PAD(12)またはPAD(16)が評価された時に読み込まれます．座標データを得る際には，あらかじめこれらを評価してから行ってください．トリガボタンの状態の入力は，ジョイスティックと同様にSTRIG関数を使用します．

3章 BASICの内部構造

BASICをより高度に利用するためには，BASICインタプリタがどのようにプログラムを管理し実行しているか，ということに関する知識が不可欠なものとなります．ここでは，そのようなBASICの内部構造について説明します．

3.1 ユーザーエリア

ユーザーエリアの下限アドレスは，MSX1ではRAM容量が8K，16K，32K，64Kといろいろであったため個々に異なっていましたがMSX2の場合RAM容量は最低でも64Kなので，ユーザーエリアの下限アドレスはつねに8000H番地となります．なお，これはBOTTOM(FC48H)の内容から知ることができます．

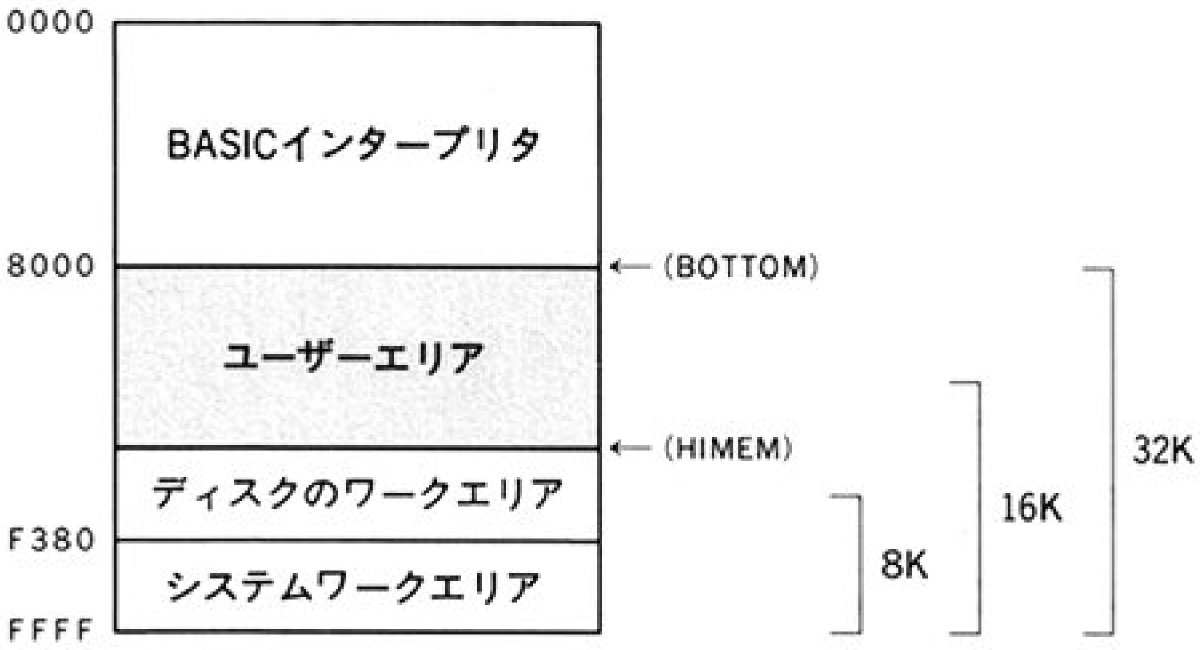
ユーザーエリアの上限アドレスは，ディスクが実装されていない場合F380H番地ですが，ディスクが実装されている場合(DISK-BASIC使用時)はドライブ数やディスク容量などにより変わってきます．これはリセット後CLEAR文を実行する前のHIMEM(FC4AH)の内容から知ることができます．主なディスク装置を実装した時のユーザーエリアの上限アドレスを**表2.19**に示します(この値は編集部独自の調査によるもので，同じ型番の製品でも値が異る可能性あり)．

ディスクドライブ名	メディア	ドライブ数	ユーザーエリアの上限
ディスクなしの状態	──	──	&HF380
ソニー(HB-701FD)	1DD	1 2	&HE68C &HE276
ソニー(HBD-30W)	2DD	1 2	&HE48D &HDE77
東芝(HX-F100)	1DD	1 2	&HE68E &HE278
ビクター(HC-F303)	1DD	1 2	&HE68F &HE279
ヤマハ(FD-05)	2DD	1 2	&HE48D &HDE77
松下(CF-3300)	2DD	1 2	&HE48D &HDE77

注）ドライブ数の"1"のものは，1ドライブ実装の時「CTRLキー」を押しながらリセットした場合．

表2.19　主なディスク装置によるユーザーエリアの違い

MSXを起動した時のメモリの状態を**図2.6**に示します.



注) RAM容量が32Kバイト以上であっても, BASICを使用している限りは32KバイトのRAMしか使用されない. ただし, MSX2においては, もう32KバイトをRAMディスクとしてBASICから使用可能となっている.

図2.6　BASICモードでのメモリの状態

なお, MSX 2でプログラムを作る場合, ユーザーエリアの上限アドレスとしては一番低い「2DD 2ドライブ」のディスク装置の実装を念頭において, 8000H番地～DE3FH番地に作成するよう推奨されています. しかし, ディスクのワークエリアがさらに大きくなる可能性もあるので, アプリケーションプログラムはかならずHIMEMの内容を確かめ, 最悪の場合でも暴走しないように対処してください. 対処の方法としては, 次のようなものが考えられます.

1. ワークエリアをリロケータブルにする
2. ワークエリアをBOTTOMから確保する
3. ドライブの数を減らすように指示して停止する

MSXはディスクが実装されている場合もSHIFTキーを押しながらリセットすることにより, ディスクを切り離すことができます. また, ディスクが1ドライブだけ実装されている時に普通に起動すると, MSXは2ドライブ分のディスクのワークエリアをとってしまうので(主に2ドライブのシミュレータに使用される), このような時はCTRLキーを押しながらリセットすることにより, 1ドライブ分のワークエリアのみ確保して起動させることが可能です. これらを行うことによって, 少しはユーザーエリアを増やすことができます.

3.2　ユーザーエリアの詳細

BASIC使用時にユーザーエリアがどのように使われているかを**図2.7**に, それらのエリアがどこから始まっているかの情報を持つワークエリアを**表2.20**に示します. なお, このワークエリアは読み出し専用となっているため(イニシャライズルーチンがリセット時に設定する), 書き換えた場合の動作に関しては保証されません.

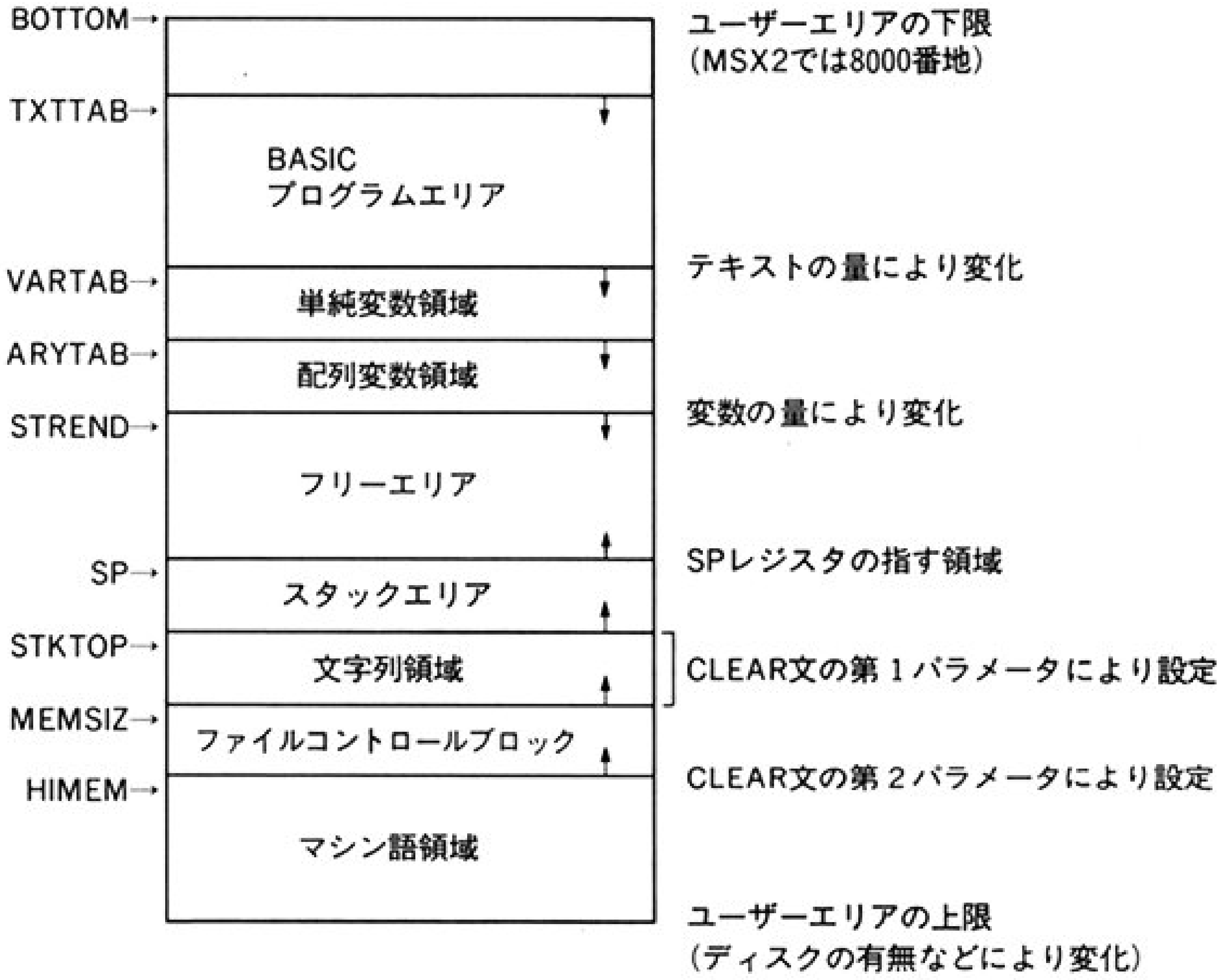


図2.7　ユーザーエリアの状態

エリア名	開始アドレス	終了アドレス
ユーザーエリア	[BOTTOM(FC48H)]	リセット時の[HIMEM(FC4AH)]－1
プログラムエリア	[TXTTAB(F676H)]	[VARTAB(F6C2H)]－1
単純変数エリア	[VARTAB(F6C2H)]	[ARYTAB(F6C4H)]－1
配列変数エリア	[ARYTAB(F6C4H)]	[STREND(F6C6H)]－1
フリーエリア	[STREND(F6C6H)]	[SPレジスタ]－1
スタックエリア	[SPレジスタ]	[STKTOP(F674H)]－1
文字列領域 (文字列領域の空エリアの先頭)	[STKTOP(F674H)] [FRETOP(F69BH)]	[MEMSIZ(F672H)]－1
ファイルコントロールブロック	[MEMSIZ(F672H)]	[HIMEM(FC4AH)]－1
マシン語領域	[HIMEM(FC4AH)]	ユーザーエリアの終わりまで

表2.20　各エリアの開始と終了アドレスを持つワークエリア

各ユーザーエリアの役割を以下に示します．

● BASICプログラムエリア

BASICで作成したプログラムは，ユーザーエリアの下限アドレス(MSX 2では8000H番地)から格納され，その大きさはプログラムの量とともに変化します．

● 変数領域

変数領域はBASICプログラムエリアのすぐ後ろに位置しており，プログラム実行時に使用された変数に関して，その名前や値の保存のため確保されます．この時の変数の格納形式を図2.8（単純変数の場合），図2.9（配列変数の場合）に示します．なお，配列変数をDIM文で宣言せずに使用すると添字が10の配列としてエリアが確保されます．ただし，4次元以上の配列はかならず宣言する必要があります．

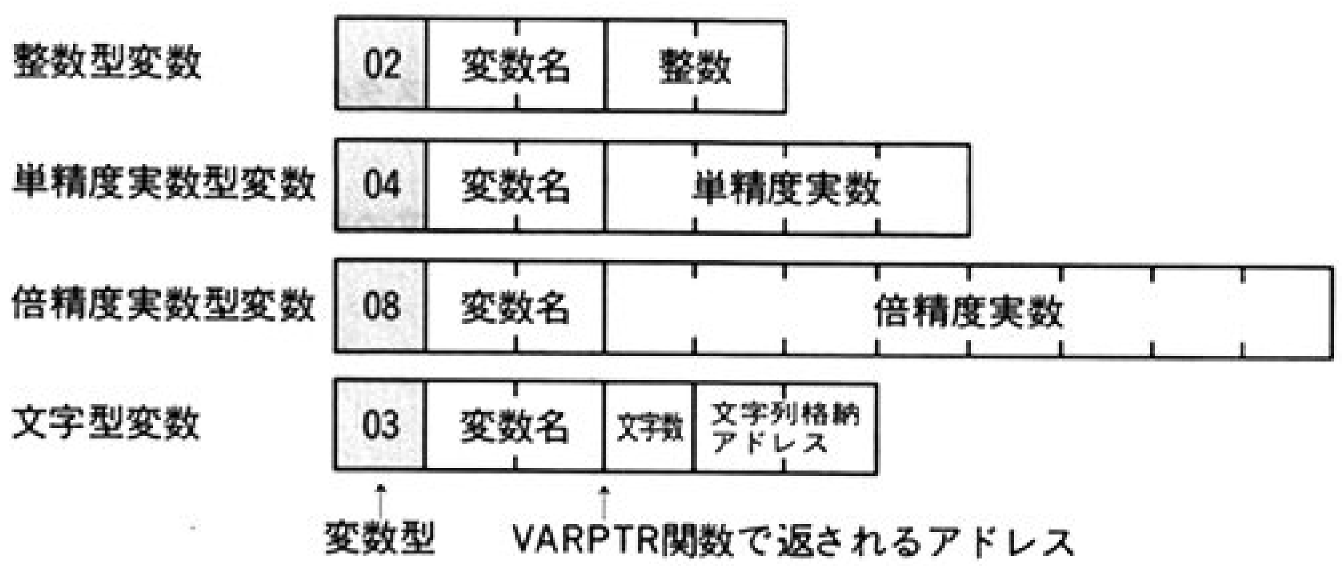
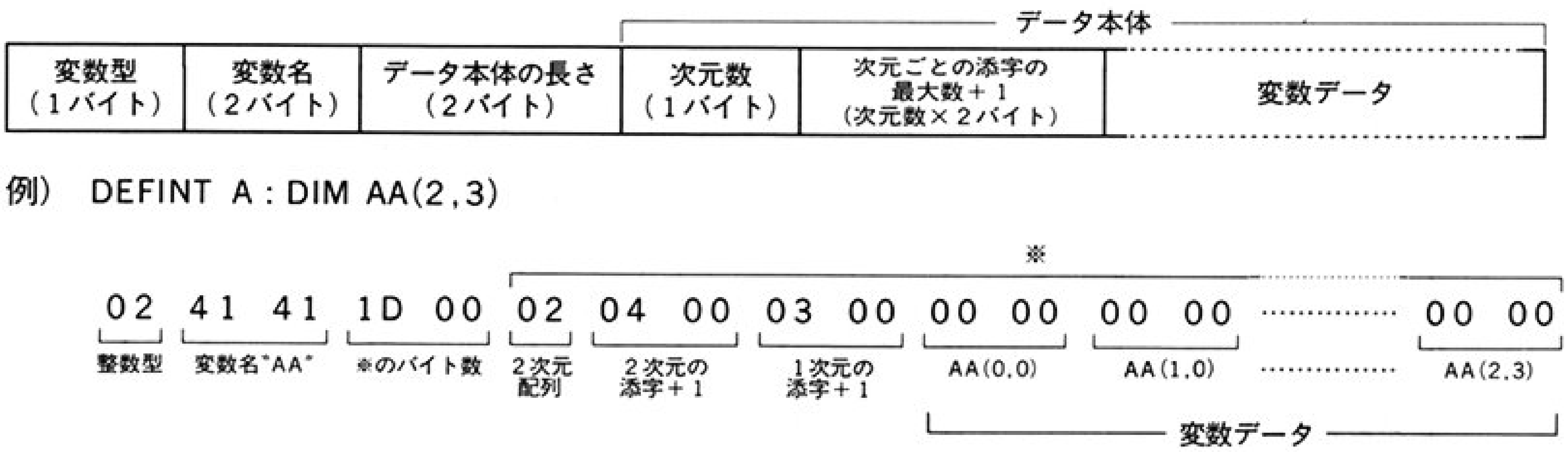


図2.8　単純変数の格納形式



注　変数データの形式は単純変数の格納形式と同じ．2バイトの数値は上位と下位バイトが逆に格納される．

図2.9　配列変数の格納形式

● フリーエリア

プログラムエリアや変数領域が大きくなったり，スタックが多くなって，フリーエリアがなくなると“OUT OF MEMORY”エラーになります．なお，BASICのFRE関数でPRINT FRE (0)を評価することにより，その大きさを知ることができます．

● スタックエリア

BASICが使用するスタックエリアです．GOSUB命令やFOR命令などの実行時に上位アドレス側から順に使用されます．

● 文字列領域

文字列変数の内容を保存するために使用される領域で，上位アドレス側から順に使用されていきます．なお，この領域の大きさはBASICのCLEAR文の第1パラメータで指定することができ，省略時は200バイトの大きさで確保されます．このエリアがすべて使われてしまうと“OUT OF STRING SPACE”となります．BASICのFRE関数でPRINT FRE（“ ”）を評価することにより，残りの領域の大きさを知ることができます．

● ファイルコントロールブロック

ファイルの情報を保存する領域で，1ファイルごとに10BH（267）バイト分の領域が取られます．ファイルいくつ分の領域を確保するかはBASICのMAXFILES文で指定でき，リセット時はファイル1つ分（MAXFILES＝1）の領域が取られています．しかし，それとは別にSAVE, LOAD命令のために，常時かならず1つ分の領域が確保されているので，実際にはこれも含めて2つ分の領域が確保されていることになります．ファイルコントロールブロックの形式を**表2.21**に示します．

変位	ラベル	意　味
＋0	FL.MOD	ファイルがオープンされたモード
＋1	FL.FCA	BDOS用FCBへのポインタ (Low)
＋2	FL.LCA	BDOS用FCBへのポインタ (High)
＋3	FL.LSA	バックアップキャラクタ
＋4	FL.DSK	デバイスナンバー
＋5	FL.SLB	インタープリタが内部で使用
＋6	FL.BPS	FL.BUFの位置
＋7	FL.FLG	いろいろな情報をもつフラグ
＋8	FL.OPS	仮想的なヘッドの位置
＋9～	FL.BUF	ファイルバッファ (256バイト)

表2.21　ファイルコントロールブロック(FCB)の形式

● マシン語領域

マシン語プログラムを扱ったりメモリを直接操作する場合の領域です．これらを行う場合は，CLEAR命令によって，あらかじめこの領域を確保しておく必要があります．

● ディスクのワークエリア

ディスクが実装時に取られるワークエリアを**図2.10**に示します．この領域はディスクが実装されていない場合はまったく存在しませんので注意してください．なお，図中右のラベルは，そこにこのアドレスの情報があることを意味します．

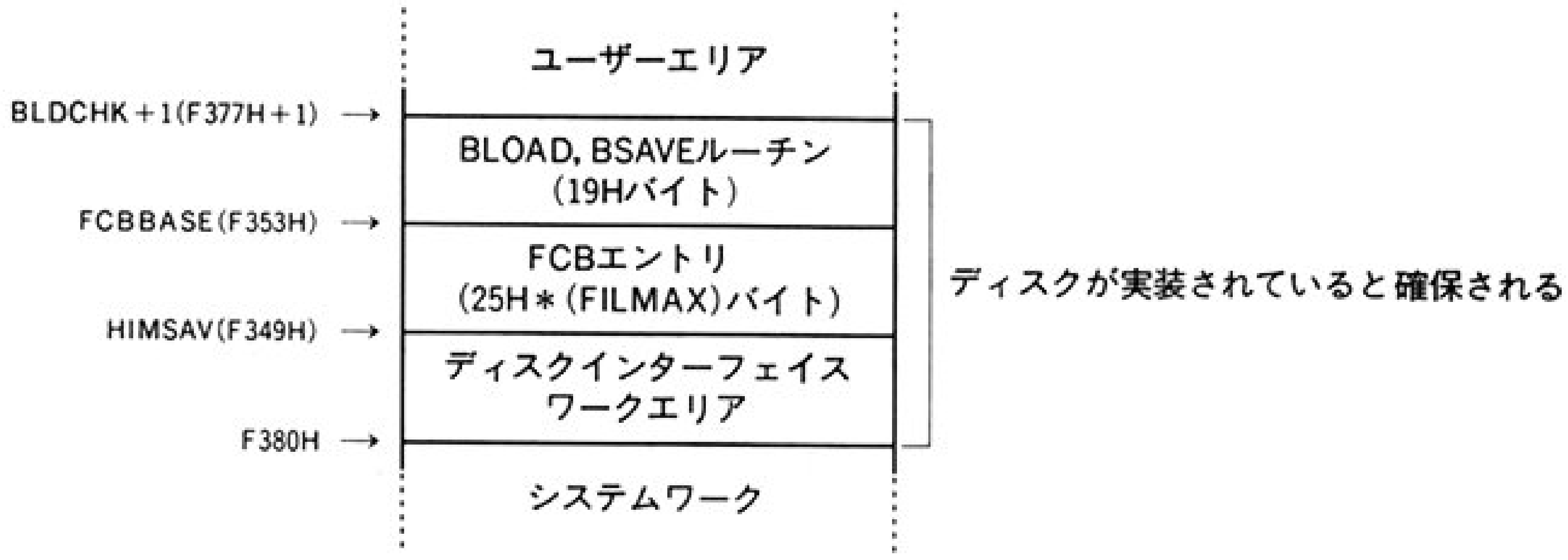


図2.10　ディスクのワークエリア

3.3　BASICプログラムの格納形式

プログラムはメモリ上に**図2.11**のような形式で記憶されており，その内容には以下のような意味があります．

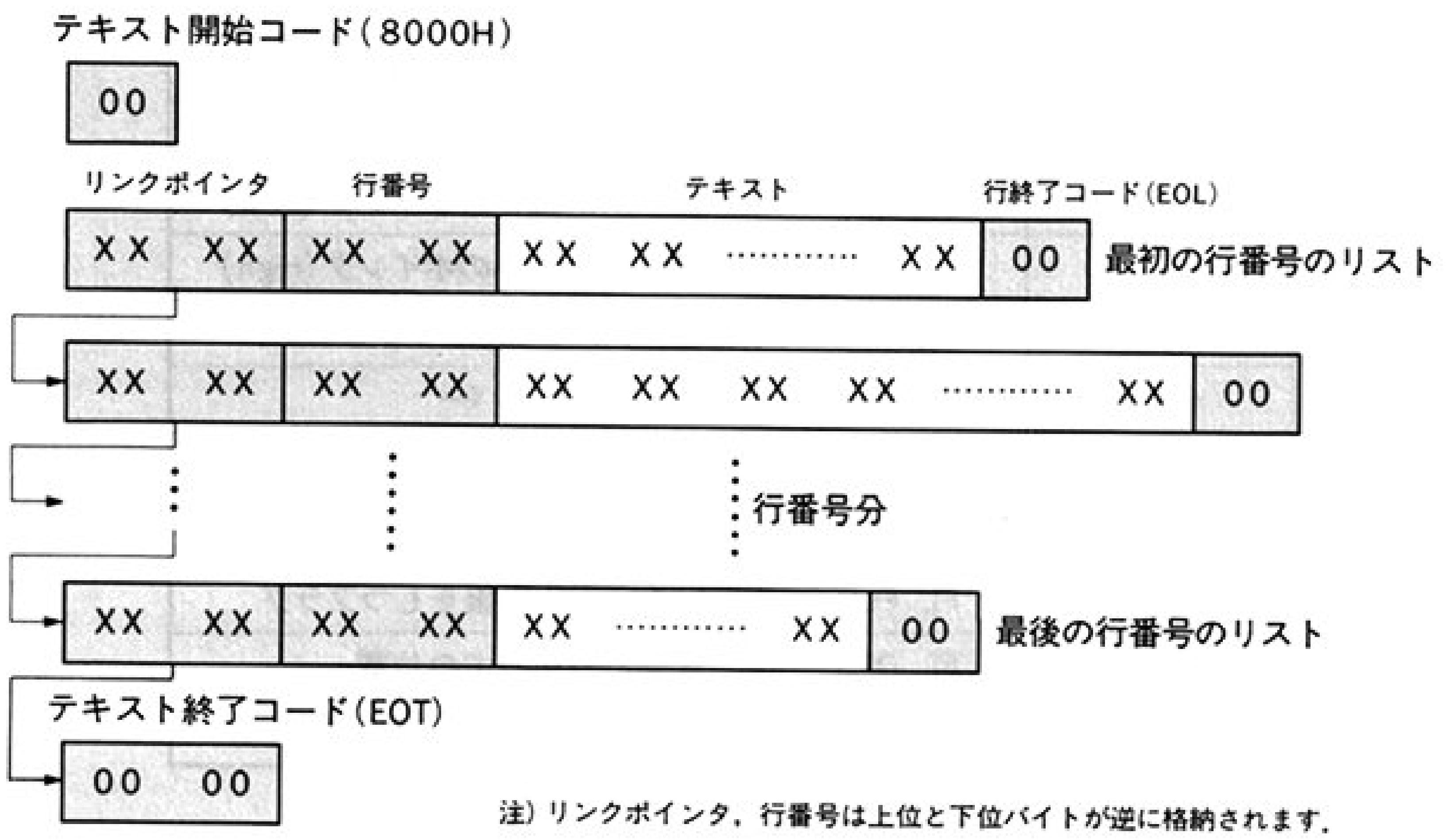


図2.11　テキスト格納形式

● リンクポインタ

次の行へのテキストポインタが絶対アドレスで書かれています．

● 行番号

文字どおりプログラムの行番号が格納されており，通常は0～65529 (0000H～FFF9H) の値が入ります．メモリを直接操作することにより65530以上の行番号を作ることも可能ですが，LISTコマンドはその行以降を表示しません．

● テキスト

ここにプログラム本体が中間コード形式で格納されています．中間コードに変換されるものは，予約語(キーワード)，演算子，数値で，その他のもの(変数名や文字列定数など)はキャラクタコードの形で格納されます．中間コードを**表2.22**に，テキスト中の数値形式を**図2.12**に示します．

キャラクタコードについては，巻末の付録を参照してください．なお，グラフィックキャラクタは「CHR$(1)+（グラフィックキャラクタコード+64)」の2バイト（2文字分）で格納されますので，グラフィック文字を取り扱う時には注意が必要です．

>	EE	CMD	D7	ERR	E2	LIST	93	PAINT	BF	SPRITE	C7
=	EF	COLOR	BD	ERROR	A6	LLIST	9E	PDL	FFA4	SQR	FF87
<	F0	CONT	99	EXP	FF8B	LOAD	B5	PEEK	FF97	STEP	DC
+	F1	COPY	D6	FIELD	B1	LOC	FFAC	PLAY	C1	STICK	FFA2
−	F2	COS	FF8C	FILES	B7	LOCATE	D8	POINT	ED	STOP	90
*	F3	CSAVE	9A	FIX	FFA1	LOF	FFAD	POKE	98	STR$	FF93
/	F4	CSNG	FF9F	FN	DE	LOG	FF8A	POS	FF91	STRIG	FFA3
^	F5	CSRLIN	E8	FOR	82	LPOS	FF9C	PRESET	C3	STRING$	E3
¥	FC	CVD	FFAA	FPOS	FFA7	LPRINT	9D	PRINT	91	SWAP	A4
ABS	FF86	CVI	FFA8	FRE	FF8F	LSET	B8	PSET	C2	TAB(	DB
AND	F6	CVS	FFA9	GET	B2	MAX	CD	PUT	B3	TAN	FF8D
ASC	FF95	DATA	84	GOSUB	8D	MERGE	B6	READ	87	THEN	DA
ATN	FF8E	DEF	97	GOTO	89	MID$	FF83	REM	3A8F	TIME	CB
ATTR$	E9	DEFDBL	AE	HEX$	FF9B	MKD$	FFB0	RENUM	AA	TO	D9
AUTO	A9	DEFINT	AC	IF	8B	MKI$	FFAE	RESTORE	8C	TROFF	A3
BASE	C9	DEFSNG	AD	IMP	FA	MKS$	FFAF	RESUME	A7	TRON	A2
BEEP	C0	DEFSTR	AB	INKEY$	EC	MOD	FB	RETURN	8E	USING	E4
BIN$	FF9D	DELETE	A8	INP	FF90	MOTOR	CE	RIGHT$	FF82	USR	DD
BLOAD	CF	DIM	86	INPUT	85	NAME	D3	RND	FF88	VAL	FF94
BSAVE	D0	DRAW	BE	INSTR	E5	NEW	94	RSET	B9	VARPTR	E7
CALL	CA	DSKF	FFA6	INT	FF85	NEXT	83	RUN	8A	VDP	C8
CDBL	FFA0	DSKI$	EA	IPL	D5	NOT	E0	SAVE	BA	VPEEK	FF98
CHR$	FF96	DSKO$	D1	KEY	CC	OCT$	FF9A	SCREEN	C5	VPOKE	C6
CINT	FF9E	ELSE	3AA1	KILL	D4	OFF	EB	SET	D2	WAIT	96
CIRCLE	BC	END	81	LEFT$	FF81	ON	95	SGN	FF84	WIDTH	A0
CLEAR	92	EOF	FFAB	LEN	FF92	OPEN	B0	SIN	FF89	XOR	F8
CLOAD	9B	EQV	F9	LET	88	OR	F7	SOUND	C4		
CLOSE	B4	ERASE	A5	LFILES	BB	OUT	9C	SPACE$	FF99		
CLS	9F	ERL	E1	LINE	AF	PAD	FFA5	SPC(	DF		

表2.22　中間コード一覧表

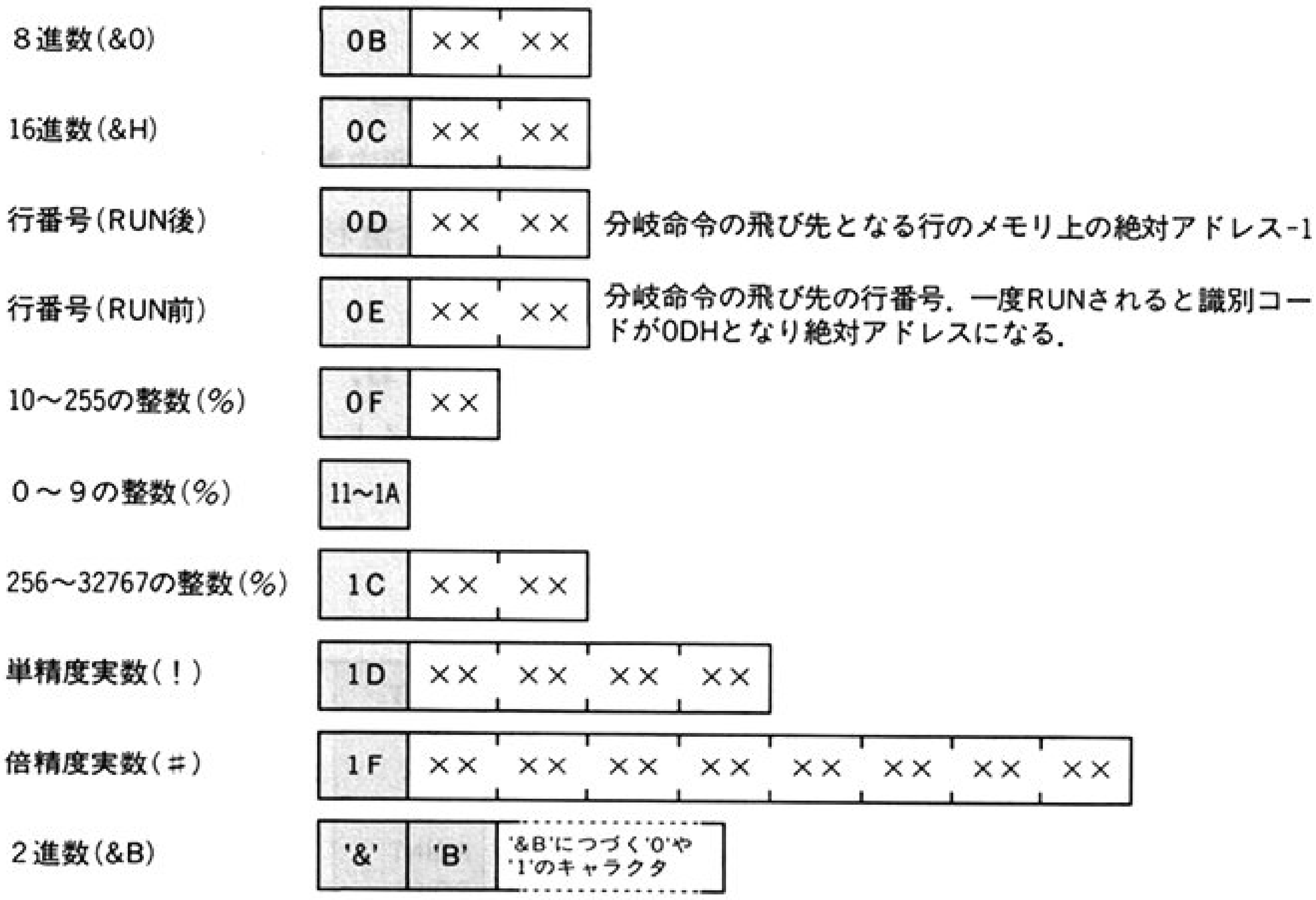


図2.12　テキスト中の数値形式

数値は予約語や変数名との区別をつけるために“識別コード”と呼ばれる番号が割り付けられており，それを見ることで後に続く値がどのようなものかわかるようになっています．

２バイトの数値の場合は，上位と下位バイトが逆に格納されます．符号付き数値については，識別コードの前に＋または－の中間コードが置かれるのみであり，数値自体はかならず正の数で記憶されます．浮動小数点表記法についてはおおむね巻末のマスパック（Mathematical Package)の説明と同じですが，上記のように数値はかならず正の数で記憶される点に注意してください．２進数（&Ｂ）については，識別コードは割り当てられておらず，アスキーコードそのままの形で記憶されます．

4章 マシン語とのリンク

これまで説明したように，MSX BASIC ver 2.0は強力な機能を備えていますが，さらに実行速度を上げたい，あるいはMSX 2のハードウェア機能をとことんまで引き出したいと思ったら，やはりどうしても多少のマシン語は使用せざるを得ません．本章では，その必要が生じた時のために，BASICからマシン語プログラムを呼び出す方法，およびその際に知っておくべきいくつかの情報について説明します．

4.1 USR関数

BASICからのマシン語の呼び出しは，以下の手順で行います．USR関数のカッコの中は，引数としてマシン語に値を渡すために用いられます．引数は数式，文字式のいずれでもかまいません．

① DEF USR文で，マシン語プログラムの実行開始アドレスを指定する．
② USR関数でマシン語を呼び出す．
③ マシン語からBASICに戻る場合は，RET命令(C9H)を実行する．

例) C000番地が実行開始アドレスであるマシン語プログラムを呼び出す．

```
DEF USR=&HC000
A=USR(0)
```

4.2 USR関数の引数と戻り値によるデータの受け渡し

BASICからマシン語に引数を渡す場合，マシン語側ではレジスタAの内容でその型を知ることができます(**表2.23**)．目的の値は引数の型により**図2.13**のような形式で格納されていますから，それに応じて値を受け取ります．例として，文字列型の引数を受け取るプログラムを**リスト2.1**に示します．

2	2バイト整数型
3	文字列型
4	単精度実数型
8	倍精度実数型

表2.23　レジスタAに代入される引数の型

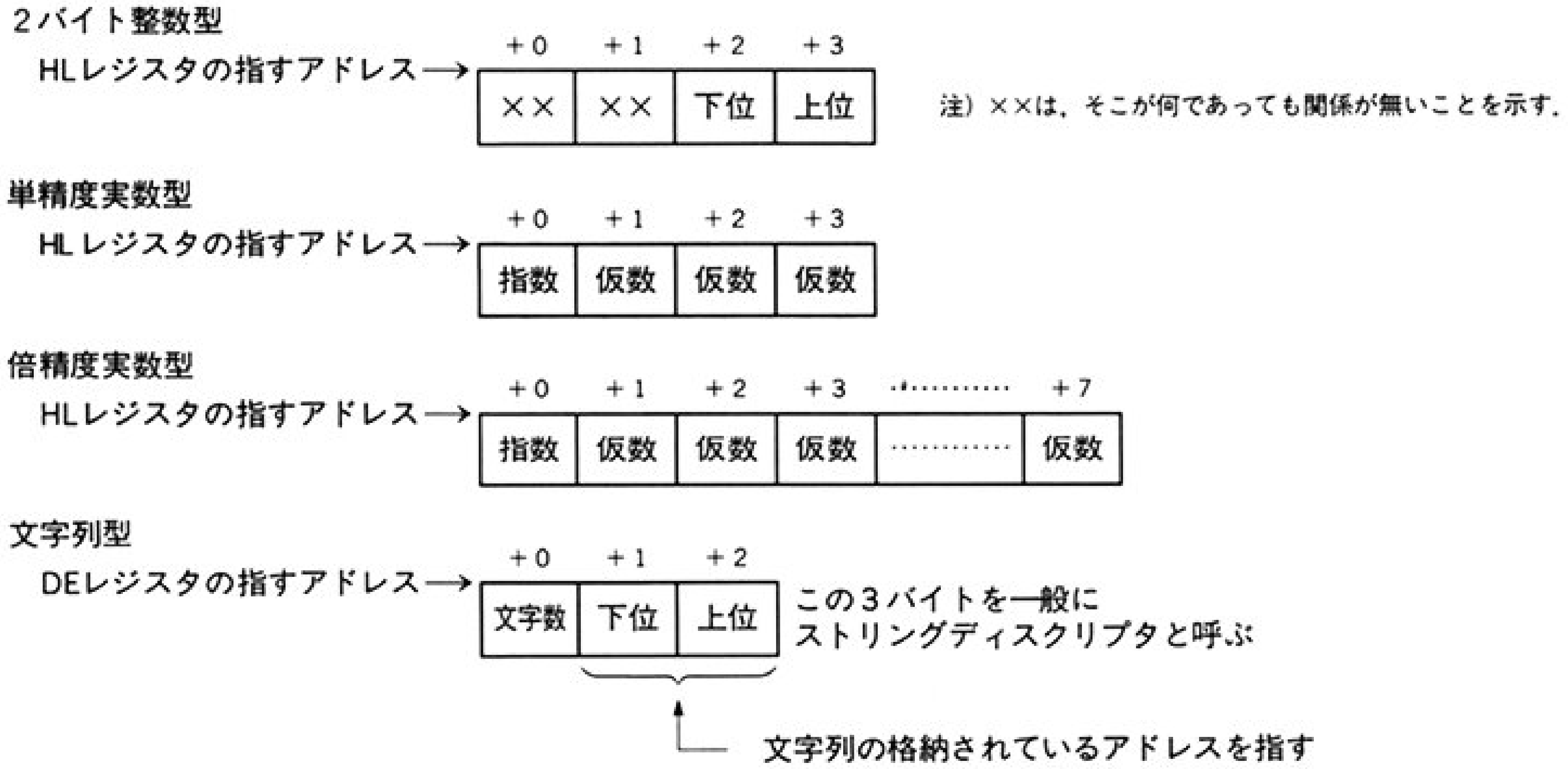


図2.13　引数による値のわたされ方

リスト2.1　文字型の引数の使用例

```
;=============================================================
;  List 2.1     print string with USR function
;               to use, do  DEF USR=&HB000 : A$=USR("STRING")
;=============================================================
;
CHPUT    EQU      00A2H             ;character output

         ORG      0B000H

RDARG:   CP       3
         RET      NZ                ;parameter is not string

         PUSH     DE
         POP      IX                ;IX := string descriptor
         LD       A,(IX+0)          ;get string length
         LD       L,(IX+1)          ;get string pointer(low)
         LD       H,(IX+2)          ;get string pointer(high)
         OR       A
         RET      Z                 ;if length=0

RD1:     PUSH     AF
         LD       A,(HL)            ;get a character
         CALL     CHPUT             ;put a character
         POP      AF
         DEC      A
         RET      Z
         INC      HL
         JR       RD1

         END
```

逆に，引数として渡されたこれらの値をマシン語の中で書き換えることによってUSR関数の値としてBASICに引き渡すことができます．この場合，VALTYP (F663H) を変更することにより戻り値をBASICからの引数の型以外のものに変更することも可能です．ただし，文字列型の場合，文字数を変えることはできません．

4.3　命令の増設

MSXでは現在“CMD”と“IPL”という予約語が未使用となっており，この予約語のフック(それぞれFE0DH，FE03H) を自分で作成した任意のマシン語ルーチンにジャンプするよう書き換えるだけで，新しい命令を作成することができます．簡単な例を**リスト2.2**に示します．

リスト2.2　CMD命令の増設

```
;==============================================================
;  List 2.2    make CMD command ( turn on/off the CAPS LOCK )
;               to initialize command:    DEF USR=&HB000 : A=USR(0)
;               to use command:           CMD
;==============================================================
;
CHGCAP  EQU     0132H           ;CAPS LAMP on/off
CAPST   EQU     0FCABH          ;CAPS LOCK status
HCMD    EQU     0FE0DH          ;CMD HOOK

        ORG     0B000H

;----- CMD initialize -----      この部分を実行するとCMDコマンドが増設される

        LD      BC,5            ;NEW HOOK SET
        LD      DE,HCMD
        LD      HL,HDAT
        LDIR
        RET

;----- new HOOK data -----       フック(FE0DH)に書き込む5バイトデータ

HDAT:   POP     AF
        JP      CAPKEY
        NOP

;----- executed by CMD -----     CMDコマンドの実体

CAPKEY: CALL    CHGCAP
        LD      A,(CAPST)
        CPL
        LD      (CAPST),A
        RET

        END
```

フックに書き込む最初の“POP AF”は，この場合“CMD”の実行の時にスタックに積まれるエラー処理アドレスを捨てるためのものです．これを入れておかないと，“RET命令”でBASICに戻るはずがエラー処理ルーチンにジャンプしてしまいます．ユーザールーチン内でエラーを出

したい場合はこのアドレスを取っておくのも1つの方法です．

なお，これらのフックは本来は将来の拡張用にリザーブしてある場所なので，商用のアプリケーション・プログラムで使用してはいけません．

4.4　CMD命令の拡張

より複雑なステートメントの拡張を行うには，CMD命令にも引数を渡せると便利です．マシン語を呼び出した際，HLレジスタがBASICテキスト中の“CMD”の次の位置を指していますから，これを利用して後に続く文字列の評価を行えば目的は果たせます．これらを行うために有効と思われる内部ルーチンを，以下に示します．

● CHRGTR（4666H／MAIN）……………テキストから1文字取り出す（図2.14）．

入力：HL←テキストを指すアドレス

出力：HL←取り出した文字のアドレス

A←取り出した文字

Zフラグ←文末（：または00H）ならばON

CYフラグ←0～9ならばON

機能：（HL＋1）の場所のテキストから1文字取り出す．スペースはスキップされる．

● FRMEVL（4C64H／MAIN）……………テキスト中の式を評価する（図2.15）．

入力：HL←テキスト中の式の先頭アドレス

出力：HL←式の次のアドレス

[VALTYP（F663H）]←型に応じて2，3，4，8の値が入る．

[DAC（F7F6H）]←式の評価結果

機能：式を評価し，型に応じて出力する．

● FRMQNT（542FH／MAIN）……………式を2バイト整数型で評価する．

入力：HL←テキスト中の式の先頭アドレス

出力：HL←式の次のアドレス

DE←式の評価結果

機能：式を評価し，整数型(INT)で出力する．結果が2バイト整数型の範囲に納まらない場合には“Overflow”エラーを発生し，BASICコマンドレベルに戻る．

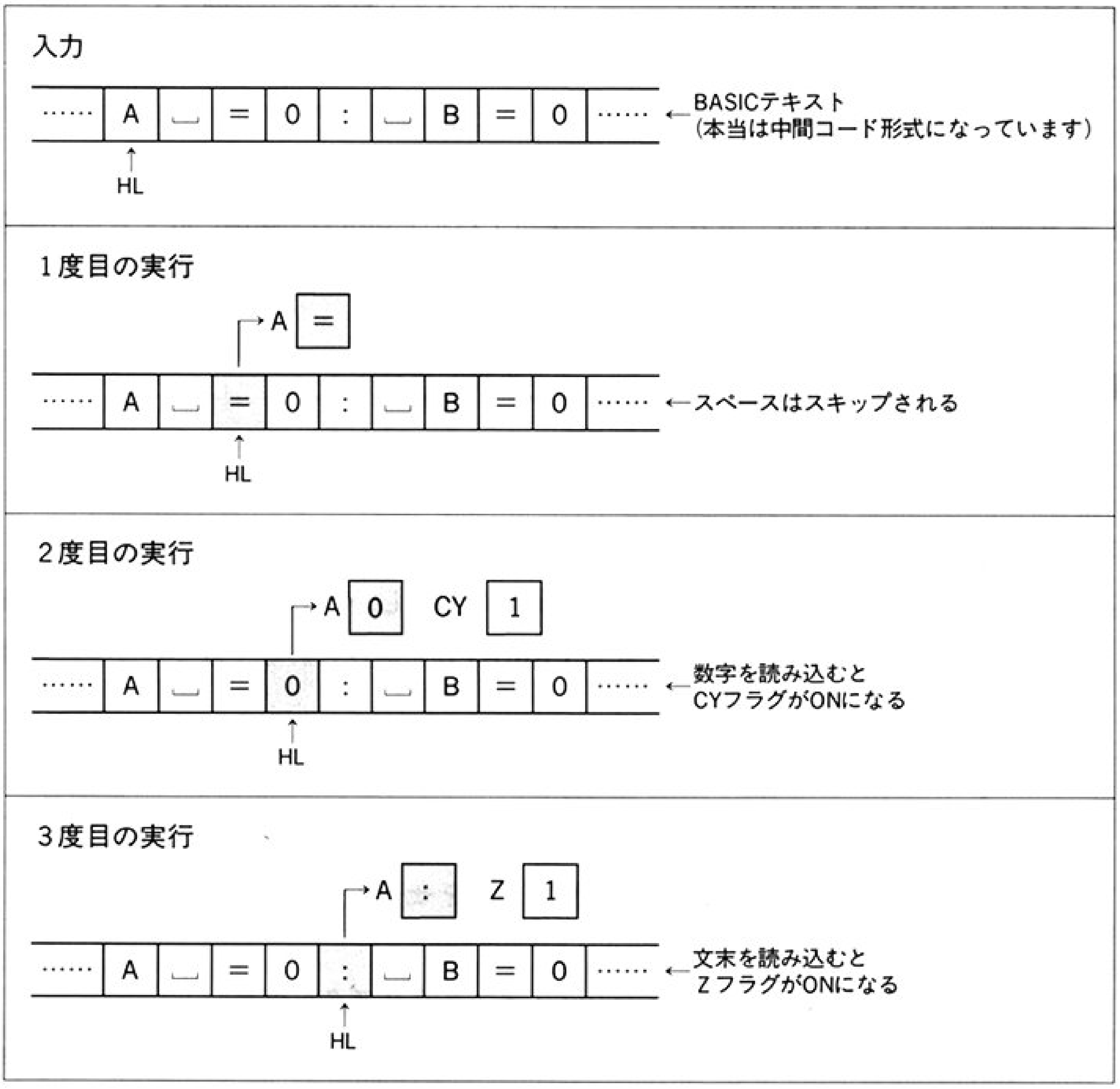


図2.14　CHRGTRの入出力状態

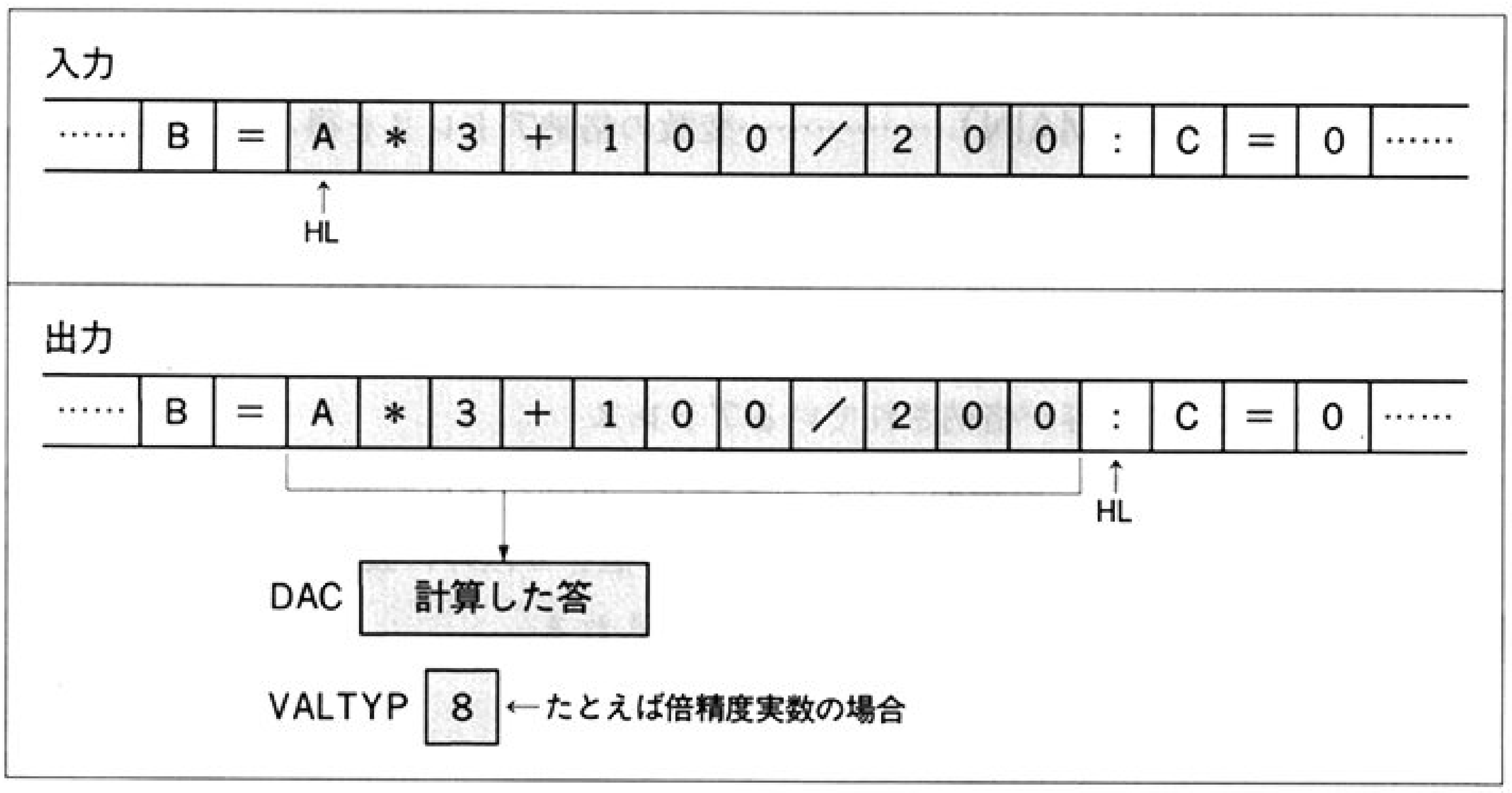


図2.15　FRMEVLの入出力状態

● GETBYT (521CH／MAIN) ……………式を1バイト整数型で評価する．

入力 HL←テキスト中の式の先頭のアドレス

出力：HL←式の次のアドレス

A，E←式の評価結果（AとEには同じ値が入る）

機能：式を評価し，1バイト型で出力する．結果が1バイト整数型の範囲に収まらない場合には“Illegal function call”エラーを発生し，BASICコマンドレベルに戻る．

● FRESTR (67D0H／MAIN) ……………文字列を登録する．

入力：[VALTYP (F663H)] ←型（文字型でないとエラーになる）

[DAC (F7F6H)] ←ストリング・ディスクリプタへのポインタ

出力：HL←ストリング・ディスクリプタへのポインタ

機能：FRMEVLで得た文字列型の結果を文字列領域に登録し，そのストリング・ディスクリプタを得る．文字列を評価した場合は前記のFRMEVLと組み合わせて以下のように使用するのが一般的である．

```
  ⋮
CALL   FRMEVL
PUSH   HL        ；テキストポインタをセーブする
CALL   FRESTR
EX     DE, HL    ；ストリング・ディスクリプタを得る
POP    HL
LD     A, (DE)   ；入力された文字列の長さを得る
  ⋮
```

● PTRGET (5EA4H／MAIN) ……………変数の格納アドレスを得る(図2.16)．

入力：HL←テキスト中の変数名の先頭アドレス

[SUBFLG (F6A5H)] ←0：単純変数，0以外：配列変数

出力：HL←変数名の次のアドレス

DE←目的の変数の内容が格納されているアドレス

機能：変数（または配列変数）の格納アドレスを得る．目的の変数の領域が確保されていなかった場合は，領域の確保も行われる．[SUBFLG]の値を0以外に設定しておくと，配列の個々の要素ではなく，その配列の先頭アドレスが得られる．

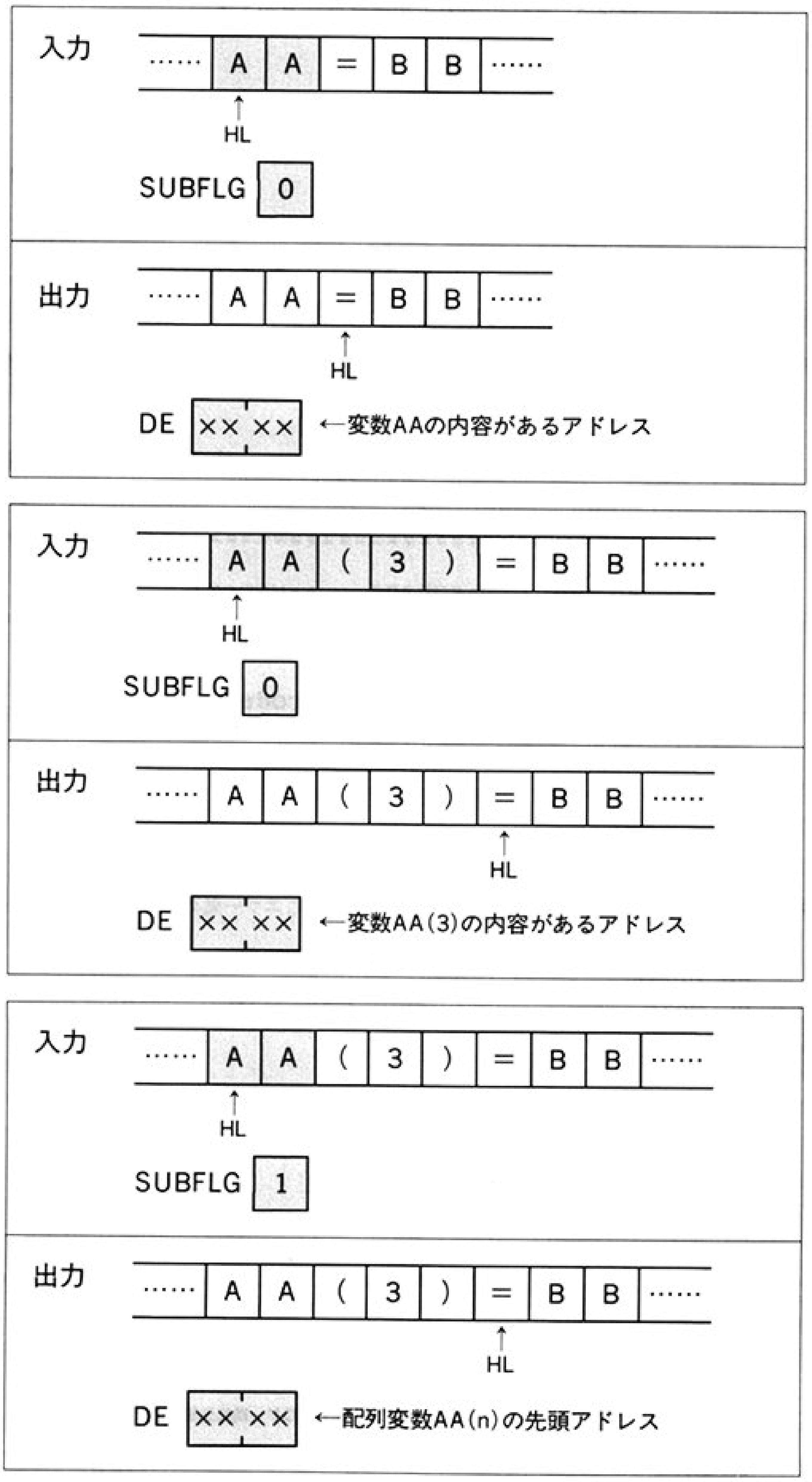


図2.16　PTRGETの入出力状態

● **NEWSTT（4601H／MAIN）**……………任意のテキストを実行する．

入力：HL←実行するテキストのアドレス

出力：——

機能：テキストの実行を行う．テキストは**図2.17**の状態となっている必要がある．

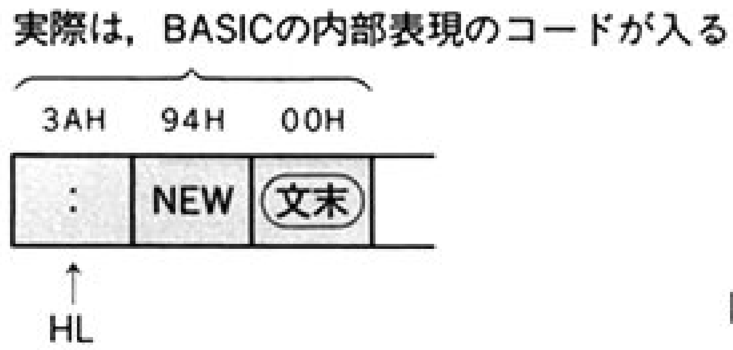


図2.17　NEWSTTのためのメモリ設定

これらの内部ルーチンは，BASICテキストを対象とするものであるため，エラーが発生した場合はBASICと同様のエラー処理が行われます．その場合，フックH.ERRO(FFB1H)の書き換えにより，ユーザーがエラー処理を行うこともできます(Eレジスタにエラー番号が入る)（リスト2.3)．

リスト2.3　エラー処理ルーチンの変更

```
;=====================================================================
;  List 2.3    Your own error
;              To use,  do DEF USR=&HB000 : A=USR(0)
;=====================================================================
;
HERR     EQU      0FFB1H          ;error hook
SYNERR   EQU      2               ;syntax err code
CHPUT    EQU      0A2H            ;character output
NEWSTT   EQU      4601H           ;run
READYR   EQU      409BH

         ORG      0B000H

;----- command initialize -----  この部分を実行するとエラー処理ルーチンが変更される

         LD       BC,5            ;SET NEW HOOK
         LD       DE,HERR
         LD       HL,HDAT
         LDIR
         RET

HDAT:    JP       ERROR
         NOP
         NOP

;----- error routine -----        エラー処理ルーチンの実体

ERROR:   LD       A,E             ;when in error, E holds error code
         CP       SYNERR          ;syntax error ?
         RET      NZ              ;no

         LD       HL,DATA1        ;yes
LOOP:    LD       A,(HL)          ;put new error message
         CP       '$'
         JR       Z,EXIT
         PUSH     HL
         CALL     CHPUT
         POP      HL
         INC      HL
         JR       LOOP

EXIT:    JP       READYR

DATA1:   DEFM     'ｷｬﾊｯ ﾏﾁｶﾞｲﾁｬｯﾀ'  ;new error message
         DEFB     07H,07H,07H,'$'

         END
```

4.5 割り込み使用法

Z80CPUにはINTとNMIの2つの割り込み端子がありますが，MSXで使用しているのはINTのみです．INT端子には60［Hz］の信号が入力されており，1秒に60回のタイマ割り込みが実行されるようになっています．Z80の割り込みモードが1に設定されているため，割り込みがかかると38H番地がコールされ，そこからタイマ割り込みルーチンへジャンプし，キー入力をはじめとする種々の処理を行っています．

このタイマ割り込みルーチンは，途中でフックH.TIMI (FD9FH) にジャンプします．このフックを利用することで，このタイマ割り込みルーチンに任意の機能を追加することが可能です．ここで注意しなければならないのは，通常ここにはRET命令が書かれているだけですが，ディスクなどの周辺機器が接続されていて，すでにこのフックが使用中の時です．この場合，知らずに書き換えるとそれら周辺機器が以後使用不能になってしまうので，どのマシンでも正常に実行できるようにするためには，あらかじめそれに対する処理をしておく必要があります．割り込みの使用例とともにこれらの対処の一例を**リスト2.4**に示します．

リスト2.4　タイマ割り込みフックの正しい使用法

```
;==============================================================
;   List 2.4    How to use HOOK safely
;               This routine uses TIMER INTERRUPT HOOK
;                                       and turn on/off CAPS LOCK
;               To start, do  DEF USR=&HB000 : A=USR(0)
;               To end,   do  DEF USR=&HB030 : A=USR(0)
;==============================================================
;
CHGCAP  EQU     0132H           ;CAP LAMP on/off
CAPST   EQU     0FCABH          ;CAP LOCK status
TIMI    EQU     0FD9FH          ;timer interrupt HOOK
JPCODE  EQU     0C3H
TIMER   EQU     020H

        ORG     0B000H

;----- interrupt on -----         フック書き換え時には以前のフックを保存する

INTON:  DI
        LD      HL,TIMI         ;OLD HOOK SAVE
        LD      DE,HKSAVE
        LD      BC,5
        LDIR

        LD      A,JPCODE        ;NEW HOOK SET
        LD      (TIMI),A
        LD      HL,INT
        LD      (TIMI+1),HL
        EI
        RET

        ORG     0B030H
```

```
;----- interrput off -----          保存しておいたフックを元に戻して終了する

INTOFF: DI
        LD      HL,HKSAVE           ;RECOVER HOOK
        LD      DE,TIMI
        LD      BC,5
        LDIR
        EI
        RET

;----- interrput routine -----

INT:    PUSH    AF
        LD      A,(CAPST)
        OR      A
        JR      Z,CAPON

CAPOFF: LD      A,(COUNT1)
        DEC     A
        LD      (COUNT1),A
        JR      NZ,FIN
        LD      A,TIMER
        LD      (COUNT1),A
        XOR     A
        LD      (CAPST),A
        LD      A,0FFH
        CALL    CHGCAP
        JR      FIN

CAPON:  LD      A,(COUNT2)
        DEC     A
        LD      (COUNT2),A
        JR      NZ,FIN
        LD      A,TIMER
        LD      (COUNT2),A
        LD      A,0FFH
        LD      (CAPST),A
        XOR     A
        CALL    CHGCAP

FIN:    POP     AF
        CALL    HKSAVE              ;old HOOK call

        RET

COUNT1: DEFB    TIMER
COUNT2: DEFB    TIMER

HKSAVE: NOP                         ;old HOOK save area
        NOP
        NOP
        NOP
        RET

        END
```

5章 ソフトウェア開発上の諸注意

MSX用のソフトウェアを開発する際，それがどのようなMSX機種上でも問題なく動作するために，ぜひとも守って頂きたい事項がいくつかあります．本章ではそれらの事項について述べるとともに，その他ソフトウェア作成時に知っていると役立つ種々の情報を紹介します．

● BIOS

BIOSを使う目的は，ハードウェアとソフトウェアを分離し，万一ハードウェアが変更されたとしてもソフトウェアをそのまま使えるようにすることです．商用のアプリケーション・プログラムで入出力処理を行う際には，かならずBIOSを使用してください(ただしVDPに関してはこの限りではない)．

BIOSはMAIN-ROMの0000H番地から始まるジャンプテーブルを通して呼び出されます．MSX2のSUB-ROMにも同様のジャンプテーブルがあり，これは拡張機能の呼び出しに使われます．ジャンプテーブルの分岐先，つまりBIOSの内容はハードウェアの変更や機能の拡張のために変更される可能性がありますので，アプリケーション・プログラムがそこを直接呼び出してはいけません．本書の中にも，BIOSのジャンプテーブル以外のアドレスを呼び出しているような例がありますが，あくまでも参考とするにとどめておいてください（APPENDIX BIOS一覧表参照)．なお，マスパックおよび前述のステートメント拡張に関する内部ルーチンは，今後もアドレスを変更しないので，アプリケーションから呼び出してもかまいません．

● ワークエリア

MAIN-RAMのF380H～FFFFH番地は，BIOSとBASICインタープリタのワークエリアですから，不用意に使わないでください．ワークエリア内の使われていない場所は将来の使用のために予約されているのでこちらも使用できません．なお，ディスクのワークエリアついては「3.1ユーザーエリア」の項を参照してください．

● RAMとスタックポインタの初期化

電源投入時のRAMの内容は不定で，システムのワークエリア以外の領域は初期化されません．ワークエリアの初期化はアプリケーション・プログラム自身で行ってください．RAMの内容

が00Hであることを期待しているために動かないソフトがあり，問題になりました．

ROMカートリッジのINITルーチン(第5部7章参照)が呼ばれた時のスタックポインタの値は不定で，ディスク・インターフェイスが先に初期化されたような場合にはそうでない場合よりもスタックポインタの値が小さくなります．スタックポインタの初期化を行わないプログラムがこのために暴走し，問題になりました．INITルーチンで起動し，そのまま処理が続く(つまりディスク等の周辺機器やBASICインタープリタを使う必要がない)プログラムはかならずスタックポインタを初期化してください．

● 拡張BIOSのワークエリア

拡張BIOSコールを使う時には，スロットが切り換えられてもCPUがワークエリアを参照できるように，スタックをC000H番地よりも上位に置いてください．同様の理由で，RS-232CのFCBも8000H番地より上位に置いてください．

● デバイスドライバなどのワークエリア

デバイスドライバやBASICから呼ばれるサブルーチンのように，他のプログラムと同時にメモリに存在するプログラムは，ワークエリアの確保に特に注意を要します．

カートリッジのINITルーチンがBOTTOM(FC48H)を書き換え，古いBOTTOMと新しいBOTTOMの間を自分のワークエリアとして予約し，さらに，そのワークエリアのアドレスをSLTWRK(FD09H)というスロット別に割り当てられた2バイトの領域に記録します．詳しくは第5部7章を参照してください．

● フック

たとえば，RS-232Cカートリッジが割り込み処理のためにフックを書き換えた場合，次のカートリッジが同じフックを書き換えるとRS-232Cのカートリッジは割り込みフックを使用することができなくなってしまいます．このようなことを防ぐために，フックを書き換える場合には以前のフックの内容(例では，RS-232Cカートリッジの割り込み処理ルーチンへのインタースロットコール命令)を別の場所にコピーしておき，自分がフックから呼ばれた際にはかならずそれを呼び出すことで，そのフックを使おうとしているカートリッジすべてに制御が渡るようにしてください(図2.18)．詳しくは第5部7章を参照してください．

● VRAMの初期化

アプリケーション・プログラム起動時のVRAMの内容は不定ですからVRAMはかならず初期化してから使用してください．

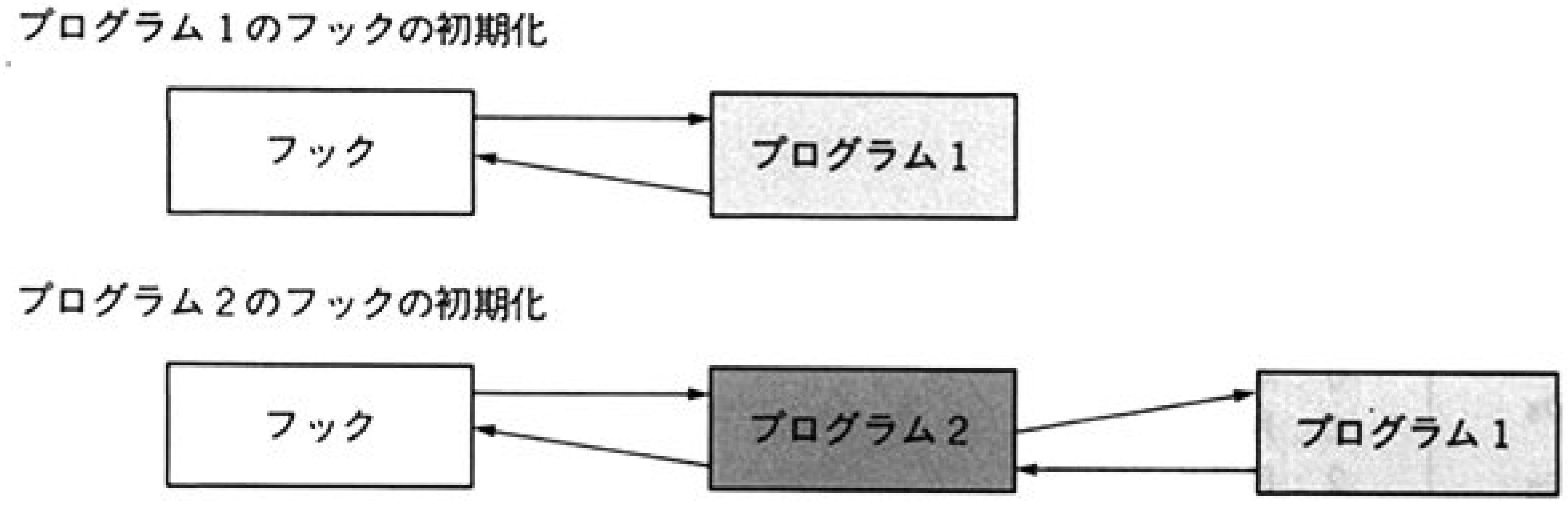


図2.18　フックの初期化

● VRAMの容量

MODE（FAFCH）のビット1，ビット2でVRAM容量を知ることができます（表2.24）．

[FAFCH]		VRAMの容量
ビット2	ビット1	
0	0	16K(MSX1)
0	1	64K(MSX2)
1	0	128K(MSX2)

表2.24　VRAM容量を知る方法

● BASICのバージョン番号

アプリケーション・プログラムがBASICのバージョン番号を知るためには，以下の方法があります．

1．MAIN-ROMの2DH番地の内容を読む（00H＝ver1.0，01H＝ver2.0）
2．ver2.0以降ではEXBRSA（FAF8H）にSUB-ROMのスロットアドレスが入っている．入っていなければ（00H）ver1.0である

● インターナショナルMSX

MSXは世界各国で使うために多くの種類がありますが，以下の項目が国により大きく異なります．

・キーボード配列，文字セット，PRINT USINGの書式
・タイマ割り込みの周波数など

IDバイトというROM内の情報を読むことで（図2.19），本体の種類がわかり，各国のMSXに対応することができます（表2.25）．

2BH

- b_0–b_3：キャラクタジェネレータの種別
  - 0：日本
  - 1：アメリカなど
  - 2：ソビエト
- b_4–b_6：日付のフォーマット
  - 0：年／月／日
  - 1：月／日／年
  - 2：日／月／年
- b_7：インターラプト周期(VSYNC)
  - 0：60Hz
  - 1：50Hz

2CH

- b_0–b_3：キーボードの種別
  - 0：日本
  - 1：アメリカなど
  - 2：フランス
  - 3：イギリス
  - 4：ドイツ
  - 5：ソビエト
  - 6：スペイン
- b_4–b_7：PRINT USINGなどの情報

図2.19 IDバイトの内容

国名	TVセット	日付フォーマット	初期画面モード	PRINT USING		
				文字列の長さ指定	文字の置き換え	通貨シンボル
日本	NTSC (60Hz)	年／月／日	SCREEN 1	&	@	¥(エン)
イギリス	PAL (50Hz)	日／月／年	SCREEN 0	＼	&	£(ポンド)
国際的	PAL (50Hz)	月／日／年	SCREEN 0	＼	&	$(ドル)
アメリカ	NTSC (60Hz)	月／日／年	SCREEN 0	＼	&	$(ドル)
フランス	SECAM(50Hz)	日／月／年	SCREEN 0	＼	&	$(ドル)
ドイツ	PAL (50Hz)	日／月／年	SCREEN 0	＼	&	$(ドル)
ソビエト	NTSC (60Hz)	月／日／年	SCREEN 0	＼	&	$(ドル)
スペイン	PAL (50Hz)	月／日／年	SCREEN 0	＼	&	$(ドル)

表2.25 国別のMSX仕様

● エスケープシーケンス

MSXにはエスケープシーケンス機能がありますが(APPENDIX参照)，これはBASICのPRINT命令，BIOSおよびBDOSコール(MSX-DOS)のコンソール出力で利用できます.エスケープシーケンスの機能は,DEC製VT52ターミナルおよびヒースキット製H19ターミナルのサブセットになっています.

● BASICに戻る方法

・ウォームスタート

MAIN-ROMのスロットを選択した後，MAIN-ROMの409BH番地へジャンプしてください．BASICのワークエリアが壊れていなければ，BASICのコマンド入力状態になります．なお，ジャンプする時のレジスタやスタックの内容はいっさい無視されます．

もう1つの方法として，内部ルーチンであるNEWSTT(第2部4.4参照)で次の命令を実行することによっても行うことができます(図2.20).

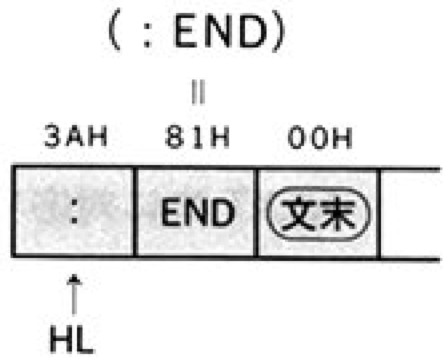


図2.20 ウォームスタートのためのNEWSTTの入力設定

● オートスタート

BIOSやBASICのワークエリアを使用しないような単純なゲームカートリッジなどではROMのヘッダの"INIT"の場所にプログラム実行開始アドレスを書くことで，そのプログラムを起動することができます．しかし，この方法を使うと他のカートリッジの初期設定が行われないために，ディスクドライブなどが使えなくなります．

それを避けるために，FEDAHに"H.STKE"というフックがありますから，カートリッジの"INIT"ルーチン実行時に起動させたいプログラムへのインタースロットコール命令をここに書き，RET命令でシステムに戻ってください．すると，すべてのカートリッジの初期設定が終わり，ディスクがあればDISK BASICの環境が整ってからそのフックが呼び出されますから，目的のプログラムを起動することができます．なお，この方法はディスクがない場合でも有効です（APPENDIX参照）．

エラーコード一覧表

1.	NEXT without FOR	NEXT 文に対応する FOR 文がない．
2.	Syntax error	構文に誤りがある．
3.	RETURN without GOSUB	RETURN 文と GOSUB 文の対応がとれていない．
4.	Out of DATA	READ 文で読むべきデータがない．
5.	Illegal function call	関数の使い方に誤りがある．数値の指定に誤りがある．
6.	Overflow	数値がオーバーフローした．
7.	Out of memory	フリーエリアがなくなった．
8.	Undefined line number	指定した行番号がプログラムにない．
9.	Subscript out of range	配列変数の添字の値が宣言された範囲を超えている．
10.	Redimensioned array	配列を二重定義しようとした．
11.	Division by zero	0 除算しようとした．0 に対して負のベキ乗を行った．
12.	Illegal direct	ダイレクトモードでは実行できないステートメントをダイレクトモードで実行しようとした．
13.	Type mismatch	データの型が一致していない．
14.	Out of string space	文字列領域がなくなった．
15.	String too long	文字列が 255 文字を超えた．
16.	String formula too complex	指定した文字列が複雑すぎる．
17.	Can't CONTINUE	CONT コマンドの実行ができる状態ではない．
18.	Undefined user function	DEF FN 文で定義されていないユーザー定義関数を使おうとした．
19.	Device I/O error	デバイスの入出力中にエラーが発生した．
20.	Verify error	カセットとメモリのプログラムが同じでない．
21.	No RESUME	エラー処理ルーチンに RESUME 文がない．

22.	RESUME without error	エラー処理ルーチン以外でRESUME文を使用した.
23.	未定義	
24.	Missing operand	必要なパラメータの指定がされていない.
25.	Line buffer overflow	入力データの文字数が多すぎる.
26. ⋮ 49.	未定義	
50.	FIELD overflow	FILED文で定義したフィールドのサイズが256バイトを超えた.
51.	Internal error	BASIC内部でエラーが発生した.
52.	Bad file number	OPENされていないファイル番号を指定した. 指定したファイル番号がMAXFILES文で指定した数を超えている.
53.	File not found	指定したファイルがない.
54.	File already open	このファイルはすでにOPENされている.
55.	Input past end	ファイルをすべて読んでしまった後，さらに読み込もうとした.
56.	Bad file name	ファイル名の指定に誤りがある.
57.	Direct statement	アスキー形式のプログラムをロード中，プログラム以外のデータがあった.
58.	Sequential I/O only	シーケンシャルファイルに対してランダムアクセスを行おうとした.
59.	File not OPEN	指定したファイルがまだOPENされていない.
60.	Bad FAT	ディスクのフォーマットが正常でない.
61.	Bad file mode	OPENされているモードに対して，正常な入出力を行っていない.
62.	Bad drive name	ドライブ名の指定に誤りがある.
63.	Bad sector number	セクタ番号に誤りがある.
64.	File still open	ファイルがクローズされていない.
65.	File already exists	NAME文で指定したファイル名が，すでにディスクに存在している.

66.	Disk full	ディスクの空き領域がなくなった．
67.	Too many files	ファイル数が 112 を超えた（ディレクトリの空きがなくなった）．
68.	Disk write protected	ディスクが書き込み禁止状態になっている．
69.	Disk I/O error	ディスクの入出力中になんらかの障害が起こった．
70.	Disk offline	ディスケットが入っていない．
71.	Rename across disk	NAME 文を異なるディスク間で行おうとした．
72. ⋮ 255.	未定義	

※ユーザーがエラー定義したい場合は，番号の大きい方から使用してください

第3部

MSX-DOS

コンピュータを単なるゲームマシンとしてではなく，何らかの目的に役立つ実用的な道具として使用するためには，大容量かつ高速にアクセス可能な外部記憶装置，すなわちディスクドライブがどうしても必要になります．そして，ディスクに蓄えられた大量のデータを効率良く操作するためには，優れたDOS（ディスク・オペレーティング・システム）の存在は欠かせません．MSXシステムのために作られたMSX-DOSは，16ビットマシンの世界で定評のあるMS-DOSの流れを汲み，Z80CPU用のDOSとしては現在最高のディスク操作環境をユーザーに提供します．MSXの実力は，このMSX-DOSを使用することによって，初めて発揮されるのだと言ってもよいでしょう．第3部では，このMSX-DOSの基本的な操作と，システムコールを通した利用法を解説します．

1章　概　要

MSX-DOSとは，いったいどのようなソフトウェアであり，ユーザーに何をもたらしてくれるのでしょうか．本章ではMSX-DOSの持つ特徴と機能，そのソフトウェア構成などの解説を通して，MSX-DOSを紹介していきます．

1.1 MSX-DOSの特徴

● ディスク環境の統一

MSX-DOSはMSXコンピュータのためのディスク・オペレーティング・システムです．MSXのすべてのバージョンに対応し，MSX 2と従来のMSXのどちらのハードウェア上でも問題なく動作可能です．また，MSXにおけるディスクの取り扱いは，かならず何らかの形でMSX-DOSを介して行われます．MSX DISK-BASICにおいても例外ではなく，そのディスク入出力には，BDOSコールと呼ばれるMSX-DOSの機能の一部が利用されています．したがってMSX-DOSとDISK-BASICには同一のディスク・フォーマットが用いられ，一部のパソコンで見られるように，BASICとDOSの間のファイル変換を行う必要はありません．このことは，MSX-DOSをMSXのプログラム開発環境として使用する場合，操作性およびファイル資源の有効利用という点で，非常に大きなメリットとなります．

● MS-DOSとの互換性

MSX-DOSは，16ビット・パソコン用のディスク・オペレーティング・システムとして作られたMS-DOS(ver1.25)をもとに作成され，MS-DOSと共通のファイル・フォーマットが用いられています．そのためMS-DOSとファイルレベルでの互換性があり，MS-DOSのディスクに書き込まれているファイルをMSX-DOSで読み書きする，逆にMSX-DOSのディスク上のファイルをMS-DOSを用いて読み書きすることが可能です．また，DOSを操作するコマンドのレベルでも両者は統一が図られているため，MSX-DOSでDOSに親しんだユーザーは，後々16ビット機を使用することになった場合，MS-DOSへの移行にとまどうことがありません．

● CP／M のアプリケーションが利用可能

さらに，CP／M-80 とシステムコールの互換性があり，CP／M 上で作成された大部分のプログラムは，1バイトも手を加えずに MSX-DOS で実行可能です．これは，CP／M 用の膨大なアプリケーションが MSX-DOS 上で手軽に利用できることを意味します．多くの優れたソフトウェアを MSX ユーザーに提供するソフトウェア・バスとしても，MSX-DOS はたいへん有効な存在となり得るのです．

1.2　MSX-DOS の使用環境

● 必要なシステム

MSX-DOS を使用するためには，最低限64Kバイトの RAM を実装した MSX 本体と，CRT，ディスクドライブ1台，そしてディスク・インターフェイス ROM が必要です．RAM が64Kバイトに満たない場合はディスクドライブが存在しても MSX-DOS は起動せず，DISK-BASIC の環境しか利用できません．MSX 2 仕様のコンピュータは，少なくとも64Kバイトの RAM を実装することが保証されていますから，かならず MSX-DOS に対応可能です．ディスク・インターフェイス ROM は，実際はつねにディスクドライブに付属して供給され，ドライブ内蔵の MSX では本体内，ディスクカートリッジ使用の場合はカートリッジ内に存在します．

● サポートするシステム

MSX-DOS は，8台までのディスクドライブをサポートします．なお，1ドライブシステムの場合，2ドライブ・シミュレーション（1台のドライブをディスケットを交互に入れ換えることで時分割的に2ドライブとして用いるもの）の機能を持ちます．周辺装置は，キーボード入力，画面出力，プリンタ出力に対応しています．

● サポートするメディア

ディスクの物理的な構造に依存しない柔軟なファイル管理機構を持つ MSX-DOS は，多種のメディアに対応可能ですが，標準として用いられるのは3.5インチ倍密度のフロッピーディスクです．このディスクには，1DD と呼ばれる片面タイプと 2DD と呼ばれる両面タイプの2種類があります．また，それぞれのフォーマット形式にも1トラックが8セクタまたは9セクタの2種類があり，都合4種類のメディアが利用可能です．これらは**表3.1**に示した形式（マイクロソフト・フォーマット）でフォーマットされます．

	1DD, 9セクタ	2DD, 9セクタ	1DD, 8セクタ	2DD, 8セクタ
メディアID	0F8H	0F9H	0FAH	0FBH
面数	1	2	1	2
1面に含まれるトラック数	80	80	80	80
1トラックに含まれるセクタ数	9	9	8	8
1セクタに含まれるバイト数	512	512	512	512
クラスタのサイズ	2セクタ	2セクタ	2セクタ	2セクタ
FATのサイズ	2セクタ	3セクタ	1セクタ	2セクタ
FATの個数	2	2	2	2
作成可能ファイル数	112	112	112	112

注）表中の言葉の意味については，3章を参照

表3.1　MSX-DOSのサポートするメディア

1.3　MSX-DOSのシステム資源

● メモリマップ

MSX-DOSを構成するソフトウェアは，COMMAND.COM，MSXDOS.SYS，ディスク・インターフェイスROMの3つの部分に分かれており，MSX-DOSが動作している時にはメモリ上で**図3.1**に示す位置を占めています．COMMAND.COMおよびMSXDOS.SYSは，MSX-DOSが起動するまではファイルとしてディスク上に存在し，MSX-DOSが起動するとRAMに読み込まれます．ディスク・インターフェイスROMは，ディスクドライバ，DOSカーネル，DISK-BASICインタープリタを含みます．

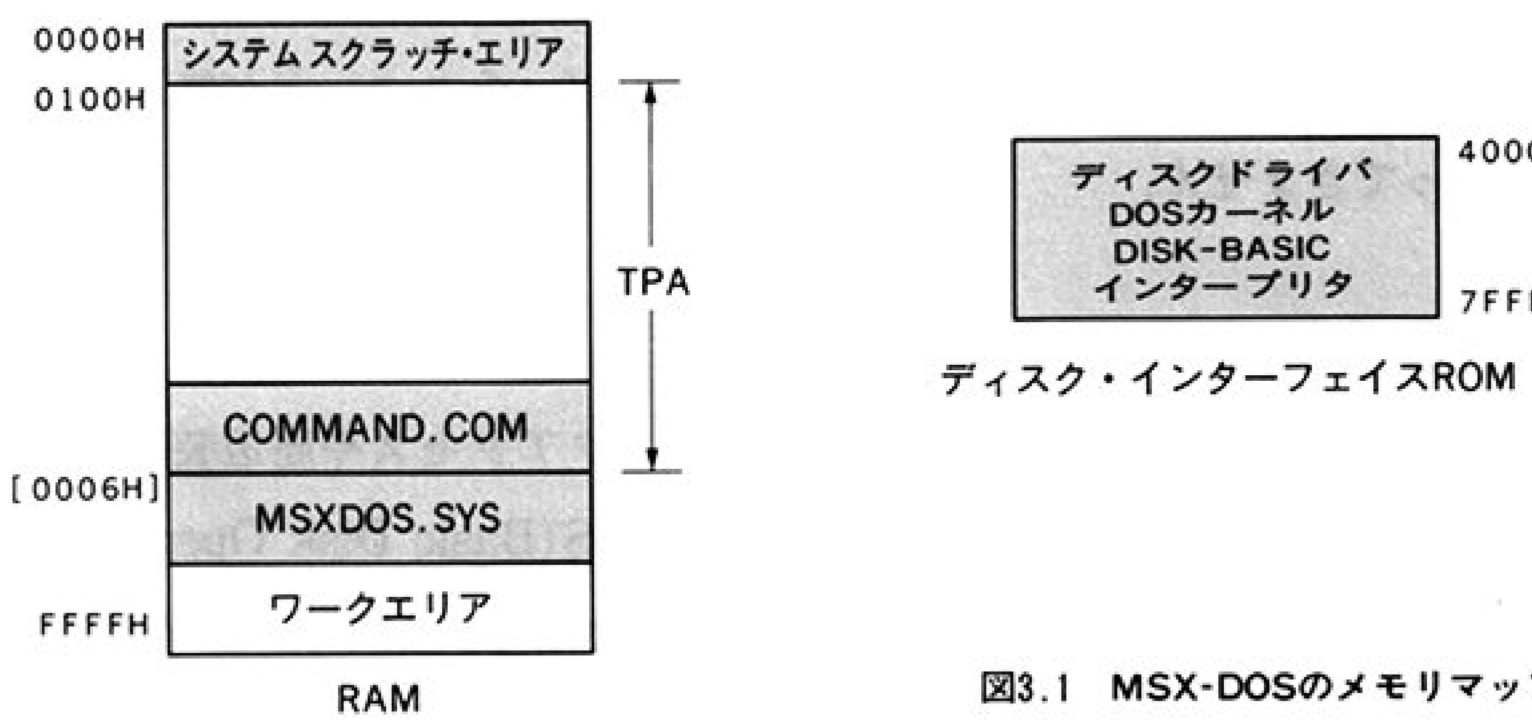


図3.1　MSX-DOSのメモリマップ

RAM上の00H～0FFH番地は，システムスクラッチ・エリアと呼ばれ，MSX-DOSが他のプログラムとのデータの受け渡しに利用する領域です．後述するシステムコールを使う場合には，この領域が重要な役割を果たします．0100H番地から始まり，RAMの0006H番地の内容で示されるアドレスまでの領域は，TPA(Transient Program Area)と呼ばれ，ユーザーが自由に利用可

能です．MSXDOS.SYSはTPAより上位アドレスに常駐し(この部分が壊れると暴走する)，COMMAND.COMはTPA内に置かれます．

● COMMAND.COM

MSX-DOSの基本的な操作は，キーボードからコマンドを入力し実行させる，という処理の繰り返しです．この時，文字列の入力からコマンドの解釈実行までの作業，いわゆるユーザー・インターフェイスに相当する部分を受け持つのがCOMMAND.COMというプログラムです．COMMAND.COMが実行するプログラムは，内部コマンド，バッチコマンド，外部コマンドの3つに分けられます．

内部コマンドはCOMMAND.COMの内部に組み込まれているものであり，RAM上に存在しています．内部コマンドが入力されると，COMMAND.COMはすぐにその部分をコールし，コマンドを実行します．

外部コマンドの場合，COMMAND.COMは，まずディスクからそのルーチンをTPAに読み込み，実行に移ります(外部コマンドの実行はつねに100H番地から始まる)．この時，COMMAND.COMは自分自身の領域を外部コマンドのために解放します．つまり，COMMAND.COMを消して，その上に外部コマンドを書き込んでしまってもかまわないのです．外部コマンドが“RET”によって終了すると，まずMSXDOS.SYSによってCOMMAND.COMが破壊されているかどうかが調べられ(チェック・サムを用いる)，そうであれば再びCOMMAND.COMをRAM上にロードしてからCOMMAND.COMに制御が移ります．

バッチコマンドは，通常はコマンドラインから行う入力を，指定されたバッチファイルから取り込むように変更することで処理されます．バッチコマンドの各ステップでは，任意の内部コマンドあるいは外部コマンドの実行が可能です．バッチコマンドの中でさらにバッチコマンドを実行することもできますが，その場合，呼び出された方のバッチコマンドが終了しても，元のバッチコマンドには戻ってきません．

● MSXDOS.SYS

MSXDOS.SYSは，MSX-DOSの中核となり，ディスクアクセスや周辺装置との入出力を管理します．MSXDOS.SYSの持つこれらの機能は，“BDOS(Basic Disk Operating System)”として，ユーザーが使用しやすいような形で公開されてもいます．公開された個々のルーチンは“システムコール”と呼ばれ，ディスクを使用するソフトウェアの開発には欠かせない存在となっています(4章参照)．ただし，それぞれの処理はMSXDOS.SYS自身によって行われているのではなく，実際の作業はDOSカーネルが受け持っています．MSXDOS.SYSは，COMMAND.COMや外部コマンドからくる入出力の要求を整理してDOSカーネルに発注する仲立ちの役割を果たしています．

なおMSXDOS.SYSの内部には，**図3.2**に示すようにBDOSの他にBIOSと呼ばれる部分が含まれています．このBIOSとは，CP/Mとの互換性のために設けられたもので，通常は使用されません．

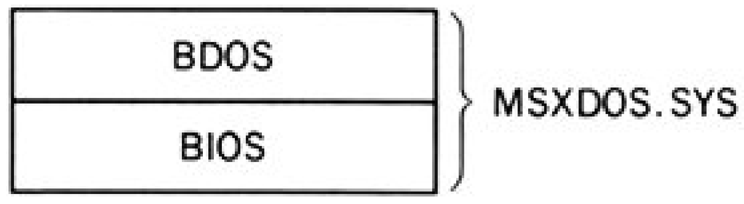


図3.2　MSXDOS.SYS

● DOSカーネル

DOSカーネルは，ディスク・インターフェイスROM中に存在する基本入出力ルーチンであり，MSXDOS.SYSのBDOS機能を実行する部分です．実際，DOSカーネルがあればシステムコールの機能はすべて実行可能です．DISK-BASICでは，直接このDOSカーネルをコールすることで，システムコールを実現しています．

● MSX-DOSの起動手順

MSX-DOSの起動は以下の過程により行われます．

1．MSXをリセットすると，始めにすべてのスロットを調べ，調べたスロットの先頭に(41H，42H)の2バイトが書き込まれていれば，そのスロットには何らかのROMが接続されていると判断します．ROMが接続されていれば，ROMのヘッダ部分にアドレスを設定されたINIT(初期化)ルーチンを実行します．ディスク・インターフェイスROMのINITルーチンの場合は，まずそのインターフェイスに接続されているドライブのためにワークエリアを確保します．

2．すべてのスロットを調べ終わったら，次にFEDAH(H.STKE)を参照します．このアドレスの内容がC9Hでなければ(INITルーチンの実行中にH.STKEのフックに対して何らかのルーチンが設定されていれば)DISK-BASICのための環境を設定し，H.STKEへジャンプします．

3．もし上記の調査でH.STKEの内容がC9Hであれば，今度はTEXTエントリを持つカートリッジを各スロットへ探しにいき，存在すればDISK-BASICのための環境を設定した後，そのカートリッジのBASICプログラムを実行します．

4．次に，ブートセクタ(論理セクタ#0)の内容がC000H～C0FFHへ転送されます．この時，“DRIVE NOT READY”または“READ ERROR”が発生した場合，あるいは転送されたセクタの先頭がEBHでもE9Hでもなかった場合，DISK-BASICが起動します．

5．次に，C01EH のルーチンがCY フラグをリセットした状態でコールされます．通常はこのアドレスには“RET NC”のコードが書き込まれているため，何も実行しないでリターンします．この部分に任意のマシン語プログラムのブートプログラムを書き込んでおけば，そのプログラムがオートスタートします．

6．次に，RAM 容量をチェックします(RAM の内容は破壊されません)．もし64Kバイトの容量がなかった場合は DISK-BASIC が起動します．

7．次に MSX-DOS の環境が初期化され，さらに CY フラグをセットした状態で C01EH がコールされます．ここでは MSXDOS．SYS が100Hからのアドレスにロードされ，100Hへジャンプします．この後 MSX-DOS は自分自身を高位アドレスに転送します．MSXDOS．SYS が存在しない場合には，DISK-BASIC が起動します．

8．MSXDOS．SYS は COMMAND．COM を 100H からのアドレスにロードし，その先頭へジャンプします．COMMAND．COM も，自分自身を高位アドレスに転送し，そこで初めて実行に移ります．COMMAND．COM が存在しない場合には，“INSERT A DISKETTE”のメッセージが表示されて，正しいディスケットがドライブにセットされるのを待ちます．

9．MSX-DOS が最初にブートされた時，“AUTOEXEC．BAT”という名前のファイルが存在すれば，それをバッチファイルとして実行します．MSX-DOS が起動せず DISK-BASIC が立ち上がった場合，“AUTOEXEC．BAS”という名前の BASIC プログラムが存在すれば，それを実行します．

2章 操　作

本章では，MSX-DOSの操作の基本となるキーボードからのコマンドライン入力の方法，およびその際に心得ておくべき事項と，MSX-DOSで利用可能な各種のコマンドの実際の使用法について説明します．

2.1 基本的な操作

● 起動時のメッセージ

MSX-DOSが起動すると，まず画面に次のようなメッセージが表示されます．

```
MSX-DOS version 1.03
Copyright 1984 by Microsoft

COMMAND version 1.11
```

図3.3 起動時の画面

上の2行は，MSXDOS.SYSのバージョンとそのコピーライトの表示です．1行おいて次の行は，COMMAND.COMのバージョンを表示しています．

● プロンプト

そしてバージョンの下に，プロンプト（入力促進記号）が表示されます．MSX-DOSのプロンプトは，デフォルト・ドライブ名 ＋ “>” の2文字で表されます．

● デフォルト・ドライブ

プロンプトの1文字目に表示されている“デフォルト・ドライブ”とは，ドライブ名を省略した際に自動的にアクセスされるドライブのことです．たとえば，デフォルト・ドライブをAとした時，Bドライブ上の“BEE”というファイルを参照するためには“B：BEE”のように表さな

ければなりません．しかし，デフォルト・ドライブAにある“ACE”というファイルは，ドライブ名を省略して単に“ACE”と表すことができます．

例1)　A>DIR B : BEE　(←Bドライブの“BEE”を参照している.)
例2)　A>DIR ACE　(←Aドライブの“ACE”を参照している.)

● デフォルト・ドライブの変更

2ドライブ以上のシステムを使用しているならば，“B：”と入力することによってデフォルト・ドライブはBに変更されます．C～Hのドライブにデフォルト・ドライブを変更する場合も同様に“C：”などと入力します．ただし，存在しないドライブを指定した場合はエラーとなります．

例1)　A>B :
　B>　(←デフォルト・ドライブがBに変更された.)
例2)　A>K :
　Invalid Drive Specification
　A>　(←Kドライブは無かった．デフォルト・ドライブは不変.)

● コマンドの入力

プロンプトが表示されている状態では，MSX-DOSはコマンドの入力待ちとなっており，ここでコマンドを入力することによって，MSX-DOSに各種の指示を与えることができます．

コマンドには**表3.2**の3形式が存在します．入力されたこれらのコマンドは，COMMAND.COMというプログラムによって解釈実行されます．MSX-DOSの操作とは，“コマンドを入力する→COMMAND.COMに実行させる”という動作の繰り返しにすぎません．

(1) 内部コマンド	COMMAND. COMの組み込みコマンド． RAM上に存在するマシン語ルーチンである． 後述の13種類が用意されている．
(2) 外部コマンド	ディスク上に存在するマシン語ルーチンである． 実行時にはディスクからロードされる． ファイル名に“COM”という拡張子を持つ．
(3) バッチコマンド	複数のコマンドを記述したテキストファイル． 記述した順にコマンドが実行される(バッチ処理)． ファイル名に“BAT”という拡張子を持つ．

表3.2　3形式のコマンド

● ファイル命名規則

MSX-DOSで扱うファイルは，以下に記述した形式の“ファイルスペック”によって表されます．

（1）ファイルスペックは「＜ドライブ＞：＜ファイル名＞」の形で表される．

（2）＜ドライブ＞はA～Hの1文字で表される．ただし，デフォルト・ドライブを指定する場合は，後ろのコロン“：”とともに省略できる．

（3）＜ファイル名＞は「＜ファイル名前部＞．＜拡張子＞」の形で表される．

（4）＜ファイル名前部＞は1文字以上8文字以下の文字列である．8文字以上指定した場合，9文字目以降は無視される．

（5）＜拡張子＞は0文字以上3文字以下の文字列である．3文字以上指定した場合，4文字目以降は無視される．

（6）＜拡張子＞は前のピリオド“．”とともに省略できる．

（7）＜ファイル名前部＞および＜拡張子＞に使用できる文字は**表3.3**のとおりである．

（8）アルファベットの大文字と小文字は区別されない．

使用可能の文字	A～Z　0～9　$　&　#　%　'　(　)　-　@ ^　{　}　~　`　! 80H～FFHのキャラクタコードに相当する文字
使用不可の文字	"　*　+　,　.　/　:　;　=　?　[　] 00H～20Hのキャラクタコードに相当する文字

表3.3　ファイル名として使用可能な文字

● ワイルドカード

ファイルスペックの＜名前＞および＜拡張子＞の記述に“ワイルドカード”と呼ばれる特殊文字を使用すると，一度に複数のファイルを指定することができます．ワイルドカードには“？”と“＊”の2種類があります．

（1）“？”は任意の1文字にあてはまります．

例）　“TEXT”，“TEST”，“TENT”　←　“TE？T”
　　　“F1−2．COM”，“F2−6．COM”　←　“F？−？．COM”

（2）“＊”は任意長の文字列にあてはまります．

例）　“A”，“AB”，“ABC”　←　“A＊”
「“COM”という拡張子を持つファイル」←　“＊.COM”
「すべてのファイル」　←　“＊.＊”

なお，実際のファイル名とワイルドカードを使用したファイル名を比較する場合，<ファイル名前部>の8文字に満たない部分および<拡張子>の3文字に満たない部分は，スペース（“ ”）で埋められているものとして考えます．このため，“A？？？.？？”という指定は，**図3.4**に示すように“ABCDE.123”とは一致しませんが，“AZ.9”とは一致してしまいます．

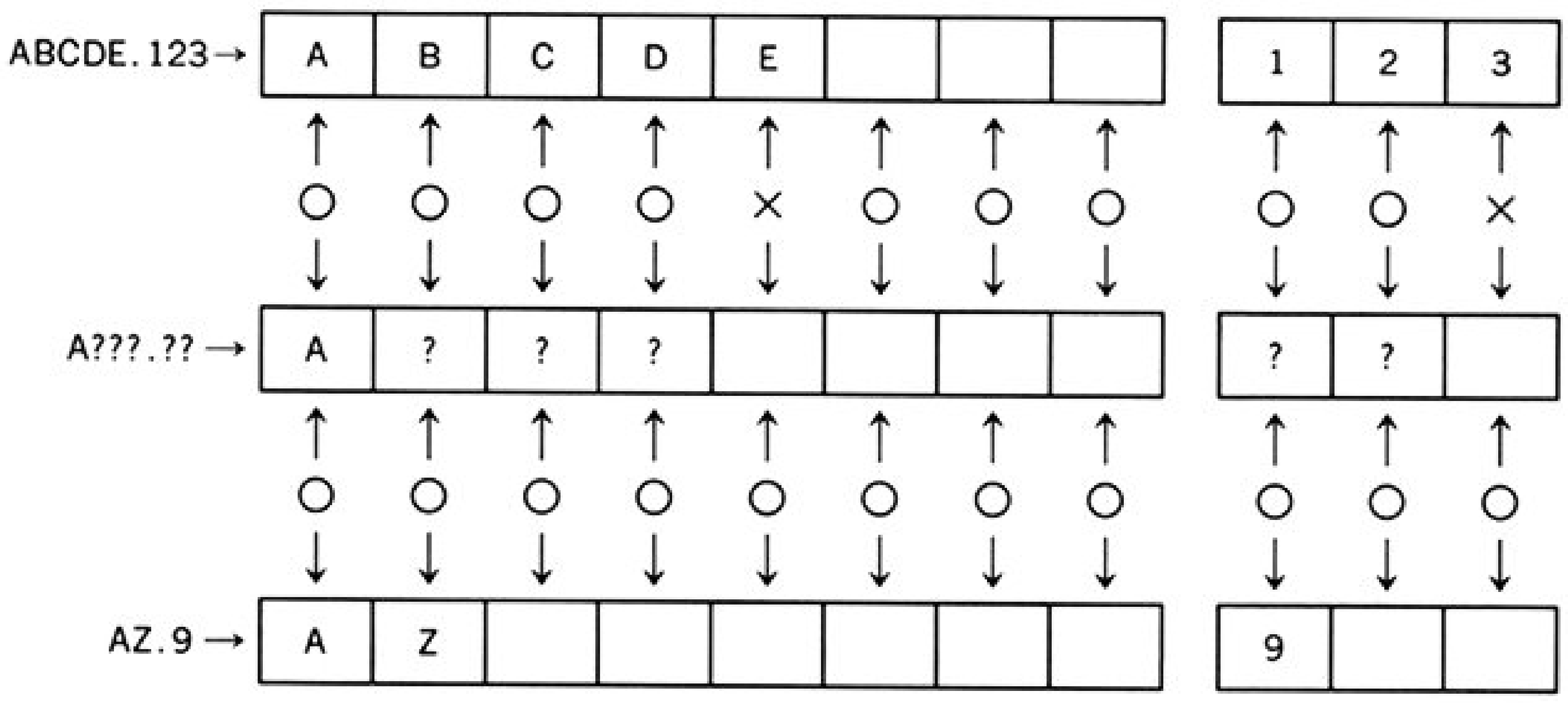


図3.4　ワイルドカードの一致

また，“＊”はいったん8個あるいは3個の“？”と置き換えられてから解釈されます．したがって，たとえば“A＊B”というファイル名は“Aで始まりBで終わる任意の文字列”とはみなされません．これは下記のように単に“Aで始まる任意の文字列”である，と解釈されてしまいます．

A＊B　（“＊”が“？”に展開される）
↓
A？？？？？？？？B　（8文字以降が削除される）
↓
A？？？？？？？

● デバイス名

MSX-DOSでは，周辺装置（デバイス）とのデータ入出力に特別なコマンドを必要としません．これは，対象となるそれぞれの装置を一種のファイル（デバイスファイル）とみなし，そのファイ

ルに対して読み書きを行うという形で入出力の動作が行われるからです．このため，MSX-DOSのユーザーは，入出力装置をディスク上のファイルとまったく同様に取り扱うことが可能です．MSX-DOSが標準でサポートしているデバイスには**表3.4**に示す5種類があり，専用のデバイス名で指定されます．ですからディスク上のファイルに，これらの名前をファイル名として使用することはできません．また，これらのデバイス名にドライブの指定や拡張子を付けたものも，単なるデバイス名と同一に扱われます．

デバイス名	指定される入出力装置
AUX	入出力拡張用の予約名，普通はNULと同じ効果を持つ．
CON	コンソール（入力として使用された場合はキーボード，出力として使用された場合は画面）．
LST	プリンタ（出力のみ，入力としては使用できない）．
PRN	プリンタ（LSTと同様）．
NUL	結果を画面やファイルに出力したくない場合にダミーとして使用する特別なデバイス．入力に用いると，かならずEOF状態になる．

表3.4　デバイス名

● テンプレートを使用した入力機能

コマンド入力時には“テンプレート”という文字バッファ領域が使用できます．テンプレートには前回入力したコマンド行がそのまま記憶され，コマンド入力時にはキーボードとテンプレートのどちらから読み込みを行うかということが1文字ごとに選択できます．このテンプレートの機能を利用すれば，前回に入力したコマンドを再度実行すること，あるいはその一部分だけを書き換えて実行することなどが簡単に行えます．テンプレートを操作するためには，**表3.6**に示したキーを使用します．

● その他の特殊キー

テンプレートを操作するもの以外に，次のコントロールキーが使用できます．これらの特殊キーの機能は，後述する一部のシステムコール使用時にもサポートされます．

	機　能
^C	実行中のコマンドを中断します．
^S	画面出力を一時停止します．
^P	画面へ表示する文字を同時にプリンタへ出力するようになります．
^N	^Pの設定を取り止め，画面出力のみとします．

表3.5　特殊キーの機能

名称	使用するキー	機能
COPY1	[→], ^¥	テンプレートから１文字入力し，コマンドラインに表示する．
COPYUP	[SELECT], ^X	次に指定する文字（キーボードにより指定）の直前までテンプレートから入力し，コマンドラインに表示する．
COPYALL	[↓], ^_	テンプレートの現在参照している位置から行末までの文字をすべて入力し，コマンドラインに表示する．
SKIP1	[DEL]	テンプレートの文字を１文字スキップする．
SKIPUP	[CLS], ^L	次に指定する文字（キーボードにより指定）の直前までテンプレートをスキップする．
VOID	[↑], [ESC], ^^, ^U, ^[	テンプレートは変えずに，現在の１行の入力をとりやめる．
BS	[←], [BS], ^H, ^]	１文字分の入力を取り消し，テンプレートの参照位置を１文字前に戻す．
INSERT	[INS], ^R	挿入モード／通常入力モードを切り換える．挿入モードでは，テンプレートの参照位置を固定したまま，キーボードからの入力をコマンドラインに表示する．
NEWLINE	[HOME], ^K	その時点のコマンドラインの内容をテンプレートに転送する．
リターンキー		その時点のコマンドラインの内容をテンプレートに転送し，それを実行する．
上記以外の通常キー		キーに相当する文字がコマンドラインに表示され，テンプレートの文字を１文字スキップする．

表3.6　テンプレートの機能

キーボード入力	コマンドライン表示	テンプレート内容（_は現在参照位置）
DIR ABCDE	A>DIR ABCDE	――――――
[RETURN]	A>	D̲IR ABCDE
[↓]	A>DIR ABCDE	DIR ABCDE_
[←][←][←]	A>DIR AB	DIR ABC̲DE
[INS] XYZ	A>DIR ABXYZ	DIR ABC̲DE
[→][→][→]	A>DIR ABXYZCDE	DIR ABCDE_
[↑]	A>	D̲IR ABCDE
[↓]	A>DIR ABCDE	DIR ABCDE_
[↑]	A>	D̲IR ABCDE
XXX	A>XXX	DIR_ABCDE
[↓]	A>XXX ABCDE	DIR ABCDE_
[HOME]	A>XXX ABCDE@	X̲XX ABCDE

表3.7　テンプレート操作の実例

● ディスクエラー

ディスクアクセス時にエラーが発生した場合，MSX-DOSは何回か再試行を試みます．それでもなおエラーが起きるならば，次のメッセージを出力し，どのような対応をするべきか質問してきます．A, R, Iの中からひとつ選んでキーを押してください．

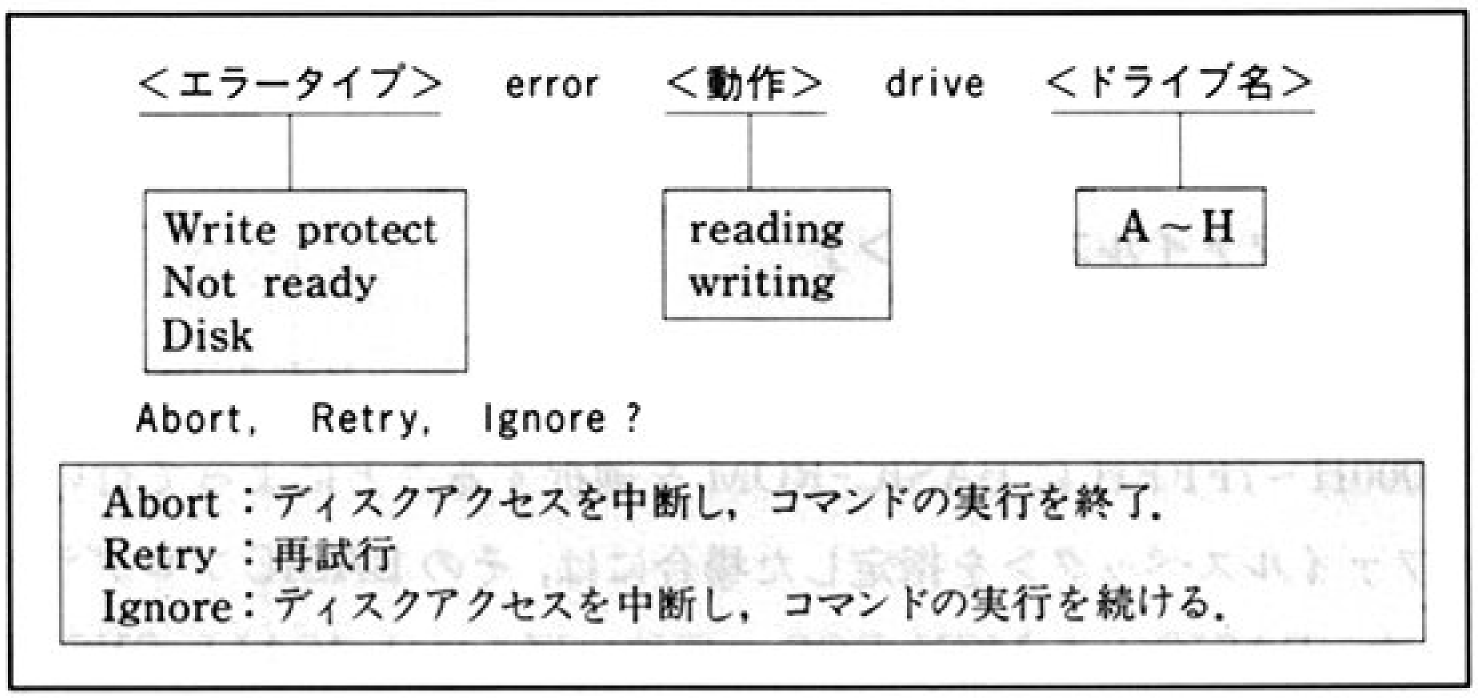


図3.5　エラーの表示

また，この形式以外に次のようなエラーが発生することもあります．これはFATの中のポインタが，存在しないクラスタを指していることを示しています．このエラーが発生した場合，そのディスケットはもう使用不可能となっています．

Bad FAT

2.2　内部コマンド

内部コマンドは，COMMAND.COMに組み込まれたマシン語プログラムです．ディスクから読み込む必要がなく，高速に実行可能です．内部コマンドには，以下の13種類が用意されています．本節では，これらのコマンドの使用法を解説します．

BASIC ……… MSX DISK BASICへジャンプする
COPY ………… ファイルをコピーする
DATE ………… 日付の表示，変更
DEL …………… ファイルを削除する
DIR …………… ファイルの一覧を表示する
FORMAT …… ディスクをフォーマットする
MODE ……… 1行に表示される文字数の変更
PAUSE ……… バッチコマンドの処理を一時停止する

REM …………　バッチコマンド中にコメント行を入れる
REN …………　ファイル名を変更する
TIME …………　時刻の表示，変更
TYPE …………　ファイルの内容を出力する
VERIFY ………　ベリファイモードのON／OFFを行う

● BASIC

書式　BASIC　[<ファイルスペック>]

DISK-BASICを起動します．これはBASICをRAM上にロードするのではなく，スロット切り換えによって0000H～7FFFHにBASIC-ROMを選択することによって行いますから，瞬時に起動します．<ファイルスペック>を指定した場合には，そのBASICプログラムを自動的に読み込み，実行します．BASICからMSX-DOSの環境に戻るには“CALL SYSTEM”を実行します．

● COPY

基本的には，あるファイルの内容を他のファイルにコピーするコマンドです．ただし，パラメータの指定法によって種々の異なった動作を行います．

(1)ファイルの複製

書式　COPY <ファイルスペック1> <ファイルスペック2>

<ファイルスペック1>で指定されるファイルを複製し，<ファイルスペック2>で指定されるファイルを作ります．ただし，ひとつのドライブ上に同じ名前のファイルを作成することはできません．ドライブが異なる場合は，同じファイル名を指定することは可能です．

使用例

A>COPY ABC XYZ　←“ABC”というファイルをコピーし，“XYZ”というファイルを作成する
A>COPY B：ABC XYZ　←Bドライブ上の“ABC”というファイルをコピーし，“XYZ”というファイルを作成する
A>COPY B：ABC C：XYZ　←Bドライブ上の“ABC”というファイルをコピーし，Cドライブ上に“XYZ”というファイルを作成する

ファイルコピーの際，アスキーモードとバイナリモードの選択が可能です．アスキーモードは“／A”スイッチで指定，バイナリモードは“／B”スイッチで指定します．デフォルトではバイナリモードが選択されます(ただし，後述のようにファイルの結合を行う場合，デフォルトの指定はアスキーモード)．両モードの違いを**表3.8**に示します．

	ソースファイルの読み出し	デスティネーションへの書き込み
アスキーモード	1AH（ファイルエンドマーク）以降は無視する	最後に1バイトの1AHを付加する
バイナリモード	物理的なファイルサイズまで読み出す	何も変更せずに書き込む

表3.8　アスキーモードとバイナリモード

使用例

A>COPY／A ABC XYZ	←ABCをXYZにコピーする(両ファイルともアスキーモード)
A>COPY ABC／A XYZ／B	←ABCをアスキーモードで読み出し，XYZにバイナリモードで書き込む

(2)他のドライブへのファイル複写

書式　COPY <ファイルスペック> [<デスティネーション・ドライブ>：]

<ファイルスペック>で指定されるファイルを，<デスティネーション・ドライブ>に同一のファイル名でコピーします．<デスティネーション・ドライブ>を省略した場合にはデフォルト・ドライブにコピーされます．なお<ファイルスペック>に含まれるドライブ名が<デスティネーション・ドライブ>と一致してはいけません．

<ファイルスペック>にワイルドカードを使用して複数のファイルをコピーすることも可能です．このような場合，ひとつのファイルがコピーされるごとに，そのファイル名が画面に表示されます．

使用例

A>COPY ＊.COM B：	←デフォルト・ドライブ上の“COM”という拡張子を持つすべてのファイルをBドライブにコピーする
A>COPY B：ABC	←Bドライブ上のABCというファイルをデフォルト・ドライブ上にコピーする

(3)多数ファイルの一括複製

書式　COPY <ファイルスペック1> <ファイルスペック2>

（<ファイルスペック1>：ワイルドカード記述　<ファイルスペック2>：ワイルドカード記述）

デスティネーションとなる<ファイルスペック2>をワイルドカードを用いて記述した場合，その中のワイルドカードキャラクタに相当する部分は<ファイルスペック1>の対応位置の文字と置き換えられます．たとえば

COPY AB−07.021 FL?X＊.V??

を実行すると，図3.6のように解釈され，“FL−X7.V21”というファイルにコピーが作成されます．

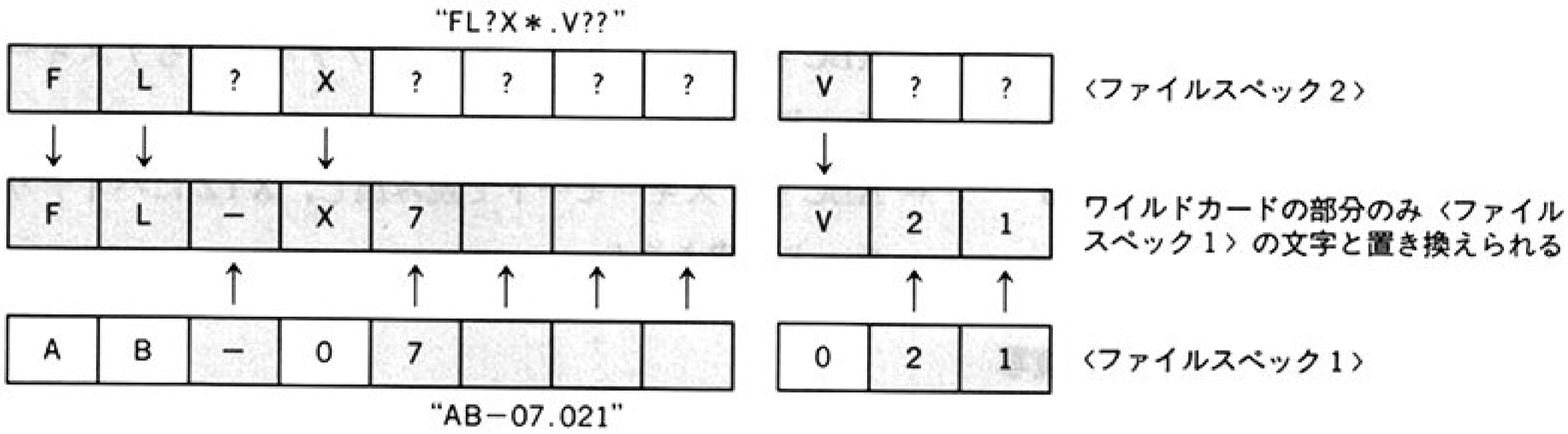


図3.6　デスティネーションファイルのワイルドカード指定

さらに<ファイルスペック1>の指定にもワイルドカードを適用すると，多数のファイルの複製を一度に作成することが可能です．

使用例

A>COPY ＊.ASM ＊.MAC　　← “ASM”という拡張子を持つすべてのファイルについて，拡張子を“MAC”に変えたものを作成する

A>COPY A＊.＊ B:Z＊.＊　　← Aで始まるファイルの先頭の文字をZに変えたものを，Bドライブに作成する

(4)ファイルの結合

書式　COPY <複数のファイルスペック> <ファイル名>

（<複数のファイルスペック>：ワイルドカードによる指定，あるいは複数のファイルスペックを“＋”で接続したもの）

ソースとなるファイルが複数であり，それを受けるデスティネーションが単独のファイル名の場合，すべてのソースファイルの内容が結合されて，指定したデスティネーションファイルに格納されます．複数のソースファイルを指定するには，ワイルドカードを用いる方法と，個々のファイルスペックをプラス記号“＋”で接続する方法があります．

ファイルの結合が行われる際，デフォルトでアスキーモードが選択され，1AHをファイルエンドマークと判断します．そのため，1AH というデータを含むバイナリファイルを COPY コマンドで結合しようとすると，最初に出てきた 1AH 以降のデータが欠落してしまいます．これを避けるためには，／Bスイッチを指定し，バイナリモードで COPY コマンドを使用してください．

また，ソースファイルの指定に２回以上ワイルドカードが現れると，２回目以降のワイルドカードは(３)の場合と同様，最初のファイル名を参照して展開されます．このことを利用すると，同じようなファイルの組の結合が一度に行えます．

使用例

A>COPY X+Y+Z XYZ	←X,Y,Z のファイルを結合し，XYZ というファイルに格納する
A>COPY ＊.LST ALL	←拡張子“LST”を持つすべてのファイルを結合し，ALL というファイルに格納する
A>COPY／B ＊.DAT ALL	←すべての“.DAT”ファイルをバイナリモードで結合する
A>COPY ASC／A+BIN／B AB／B	←ASC というアスキーファイルと BIN というバイナリファイルを結合して AB に格納する
A>COPY ＊.LST+＊.REF ＊.PRN	←ファイル名前部が等しい，拡張子“LST”のファイルと，拡張子“REF”のファイルを結合して，拡張子“PRN”のファイルを作成する

● DATE

書式　DATE ［<年>－<月>－<日>］

“／”，“.”も可

内蔵の CLOCK-IC に日付を設定します．CLOCK-IC を持たない MSX の場合は，特定のワークエリアに書き込まれます．MSX-DOS でファイルを作成あるいは変更すると，ここで設定した

日付情報がファイルごとに記録されます．

＜年＞／＜月＞／＜日＞の指定を省略してDATEコマンドを実行した場合には，現在設定されている日付が下図のように表示され，さらに新しい日付の入力待ちとなります．ここでRETURNキーだけを押せば日付は変更されません．

```
Current date is ＜曜日＞　＜年＞－＜月＞－＜日＞
Enter new date：
```

DATEコマンドで設定する日付のフォーマットは，＜年＞，＜月＞，＜日＞の3つのフィールドを“－”または“／”または“．”で区切って並べたものです．それぞれのフィールドには，次のような数値を与えることができます．

＜年＞：1980～2079(西暦年数)
　　　　0～79(2000年～2079年とみなされる)
　　　　80～99(1980～1999年とみなされる)
＜月＞：1～12
＜日＞：1～31

なお海外バージョンのMSX-DOSでは，日付のフォーマットが異なり，＜月＞－＜日＞－＜年＞または＜日＞－＜月＞－＜年＞の順で表します．

● DEL

書式　DEL ＜ファイルスペック＞
　　　　ERASEでも可

指定されたファイルを削除します．＜ファイルスペック＞にワイルドカードを用いて複数のファイルを指定することもできます．

“DEL　＊．＊”を実行するとディスケット上のすべてのファイルが削除されるわけですが，あまりにも危険な操作のため，この場合に限り確認を求めてきます．

```
A>DEL ＊．＊
Are you sure (Y／N)？
```

ここで“Y”または“y”を押すと，全ファイルが削除されます．

なお，“ERASE”によってもDELコマンドと同じファイル削除の操作が可能です．

● DIR

書式　DIR ［＜ファイルスペック＞］ ［／W］ ［／P］

＜ファイルスペック＞で指定したファイルに関して，

＜ファイル名＞ ＜ファイルサイズ＞ ＜作成日＞ ＜作成時＞

の情報を，左から順に1行に表示します．これらの情報が1行に収まらない場合には，右端に近い項目の表示は省略します．

＜ファイルスペック＞の指定には，通常のワイルドカードに加えて，以下に示す略記法が使えます．

略記法		正式な記法
DIR	=	DIR　＊.＊
DIR　＜ドライブ＞：	=	DIR　＜ドライブ＞：＊.＊
DIR　＜ファイル名前部＞	=	DIR　＜ファイル名前部＞.＊
DIR　.＜拡張子＞	=	DIR　＊.＜拡張子＞

／Wスイッチを指定すると，＜ファイル名＞のみを1行に詰めて表示します．／Pスイッチを指定すると，ファイルの数が多くて画面がスクロールしてしまう場合に，画面が一杯になった時点でいったん表示を停止し，任意のキーを押すまで待ってくれるようになります．

使用例

A>DIR	←Aドライブ上のすべてのファイルの情報を表示する
A>DIR B：	←Bドライブ上のすべてのファイルの情報を表示する
A>DIR TEST	←“TEST”という＜ファイル名前部＞を持つすべてのファイルの情報を表示する
A>DIR ／W	←Aドライブ上にあるすべてのファイル名を表示する

● FORMAT

書式　FORMAT

ディスケットを，MSX-DOS用にフォーマットします．すなわち，ディレクトリとFATを初期化し，すべてのファイルを消去します．MSX-DOSはMS-DOSと同一のディスクフォーマットを持っていますから，ここでフォーマットしたディスケットはMS-DOSからも自由に書き込み／読み出しが可能です．

FORMAT コマンドを実行すると,

Drive name？　(A,B)　　　　(←ドライブの数によって異なる)

のように，フォーマットしたいディスクの入っているドライブ名を質問してきます．これに“A”または“B”と答えると，片面フォーマットと両面フォーマットを選択できるドライブを使用している場合はメニューが表示されます．そこでフォーマットの種類を指定すると，

Strike a key when ready

と表示してキー入力待ち状態になります．ここで任意のキーを押すとフォーマット作業が始まります．フォーマットのメニューについては，ディスクドライブの説明書をご覧ください．

● MODE

書式　MODE <1行の文字数>

画面の1行に表示される文字数を設定します．<1行の文字数>には，1～80の値が設定でき，その値に応じてスクリーンモードも変わります．

<1行の文字数>	スクリーンモード
1～32	GRAPHIC1(SCREEN 1)
33～40	TEXT1(SCREEN 0:WIDTH 40)
41～80	TEXT2(SCREEN 0:WIDTH 80)

● PAUSE

書式　PAUSE [<コメント>]

MSX-DOS には，テキストファイルに記述された一連のコマンドを自動的に実行する“バッチ処理”の機能があります．このバッチ処理の実行中に，ディスクの差し替え操作などのためコマンド処理の流れを一時停止させたいことがあります．PAUSE コマンドは，そのような場合に使用します．

このコマンドが実行されると，まず画面に

Strike a key when ready...

と表示されて，キー入力待ちとなります．ここでCtrl-C 以外の任意のキーを押せば，PAUSE コマンドは終了し，作業が先へ進みます．Ctrl-Cを押すと，バッチ処理を中断します．“PAUSE”

の後ろにはどんなコメントを書いてもかまいません．これを利用して，何のためにキー入力待ちとなったのか表示することもできます．

● REM

書式　REM　[<コメント>]

バッチコマンド中にコメントを記述するために用いられます．コマンドとして何も実行しません．なお，“REM”と＜コメント＞の間はスペースを空けなければいけません．

● REN

書式　REN <ファイルスペック> <ファイル名>
　　　RENAMEも可

＜ファイルスペック＞で指定したファイルの名前を変更する命令です．＜ファイルスペック＞，＜ファイル名＞ともに，ワイルドカードを使用することができます．＜ファイル名＞をワイルドカードで指定した場合には，ワイルドカード部分は＜ファイルスペック＞の対応する位置の文字に置き換えられます(COPY コマンド参照)．

すでに存在するファイルと同じ名前に変更しようとするとエラーになります．

使用例

A>REN　ABC　XYZ　←“ABC”というファイル名を“XYZ”に変更する
A>REN　B:ABC　XYZ　←Bドライブ上の“ABC”というファイル名を“XYZ”に変更する
A>REN　*.BIN　*.COM　←ファイルの拡張子が“BIN”であるものを，すべて“COM”に変更する

● TIME

書式　TIME　[<時> [：<分> [：<秒>]]]

内蔵のCLOCK-ICに時刻を設定します．CLOCK-ICを持たないMSXの場合は，何も起こりません．MSX-DOSでファイルを作成あるいは変更すると，ここで設定した時刻情報がファイルごとに記録されます．

時刻の指定を省略してTIMEコマンドを実行した場合には，現在設定されている時刻が次のように表示され，さらに新しい時刻の入力待ちとなります．この時RETURNキーだけを押せば時

刻は変更されません．

Current time is　<時>：<分>：<秒>．<秒／100><pまたはa>
Enter new time：

TIMEコマンドで設定する時刻のフォーマットは，<時>，<分>，<秒>の3つのフィールドを"："で区切って並べたものです．<分>以降あるいは<秒>以降は省略可能あり，省略すると0とみなされます．それぞれのフィールドには，次のような数値を与えることができます．

<時>：0～23
　　12A　　　　（0時とみなされる）
　　0A～11A　　（0時～11時とみなされる）
　　12P　　　　（12時とみなされる）
　　1P～11P　　（13時～23時とみなされる）
<分>：0～59
<秒>：0～59

使用例

```
A>TIME  12          ←時刻を12：00：00に設定
A>TIME  1:16P       ←時刻を13：16：00に設定
```

● TYPE

書式　TYPE <ファイルスペック>

<ファイルスペック>で指定したファイルの内容を画面に表示するためのコマンドです．<ファイルスペック>にワイルドカードを使用すると，それに一致する最初のファイルを表示します．このコマンドは，アスキーファイルの表示を目的としたものであり，バイナリファイルを表示すると画面におかしなコントロールキャラクタが送られてしまいます．

● VERIFY

書式　VERIFY {ON ¦ OFF}

ベリファイモードの設定／解除を行います．ベリファイモードをONに設定しておくと，ディスクへのデータ書き込みの後にはかならずそれを読み出し，書き込みが正しく実行されたことをチェックします．ただし，その分ディスクアクセスに要する時間は長くなります．デフォルトでは，"VERIFY OFF"に設定されています．

2.3 バッチコマンドの使用法

MSX-DOSには，いくつかのコマンドを操作手順にしたがって並べておき，その流れに沿って自動的に一連の処理を実行させる機能があります．この操作手順が書かれているファイルを“バッチファイル”と呼び，バッチファイルによって定義される一連の操作を“バッチコマンド”と呼びます．

バッチファイルには，拡張子“BAT”が付けられます．そしてそのファイル名(から拡張子“BAT”を除いた部分)をコマンドラインから入力すると，MSX-DOSはファイルから1行ずつコマンドを読み込み，次々と実行していきます．

たとえば次のような処理を考えてみましょう．

1．Aドライブ上の“COM”という拡張子を持ったファイルを，すべてBドライブにコピーする．
2．Bドライブに最終的に集まった“.COM”ファイルの一覧表を表示する．
3．Aドライブの“.COM”ファイルをすべて削除する．

この処理は次のようなコマンドをMSX-DOSに順に与えることで実現できるはずです．

```
A>COPY  A:*.COM  B:
A>DIR  B:.COM/W
A>DEL  A:*.COM
```

もし，この3行をまとめて“MV.BAT”というバッチファイルを作成しておけば，コマンドラインから“MV”と入力するだけで自動的に目的の処理が実行できることになります．以下にその実行例を示します．

```
A>COPY  CON  MV.BAT      ┐
COPY  A:*.COM B:         │
DIR  B:.COM/W            ├ “MV.BAT”の作成
DEL  A:*.COM             ┘
^Z                         Ctrl-Z＋リターンキーの入力
A>TYPE  MV.BAT           ┐
COPY  A:*.COM B:         │
DIR  B:.COM  /W          ├ “MV.BAT”の確認
DEL  A:*.COM             ┘
```

A>MV	バッチコマンド "MV" を起動
A>COPY A：＊.COM B：	1行目の自動読み込みとその処理
⋮	
A>DIR B：.COM	2行目の自動読み込みとその処理
⋮	
A>DEL A：＊.COM	3行目の自動読み込みとその処理
⋮	

バッチコマンドの途中で Ctrl-C が押された場合，その時点で実行していた行の処理は中断され，さらに画面に図3.7のように表示されて入力待ちになります．

```
Terminate batch file (Y/N)?
```

図3.7　バッチ処理の中断

ここで "Y" を選択するとバッチコマンド全体が終了して MSX-DOS に戻り，"N" を選ぶとバッチファイルの次の行が読み込まれて，バッチコマンド自体の処理は続けられます．

● バッチ変数

バッチコマンドをより柔軟に使用するため，バッチコマンドにはコマンドラインから任意の文字列をパラメータとして渡すことができます．渡されたパラメータは，バッチコマンドの中で，"%n（nは0～9の数字1文字）" の記号で参照されます．この "%n" をバッチ変数と呼びます．

バッチ変数%1，%2，……は，コマンドラインで指定したパラメータと左から順に対応し，%0はバッチコマンド自身の名前に対応します．

```
A>COPY CON TEST.BAT ……………バッチコマンドの作成
REM %0 %1 %2 %3
^Z
        1 file copied
A>TYPE TEST.BAT
REM %0 %1 %2 %3 ……………3つの引数を表示するバッチコマンド

A>TEST ONE TWO THREE FOUR ……………バッチコマンドに引数を与えて実行
A>REM TEST ONE TWO THREE
A>
```

図3.8　バッチ変数の使用例

● AUTOEXEC.BAT

“AUTOEXEC.BAT”という名前のバッチファイルは，MSX-DOSでは特別にMSX-DOS起動時のオートスタートプログラムとして使用されます．MSX-DOSが起動した時，COMMAND.COMはこのAUTOEXEC.BATファイルが存在するかどうかを調べ，もし存在すれば実行します．

2.4 外部コマンド

外部コマンドは拡張子“COM”を持つファイルの形でディスケット上に存在し，この外部コマンドの名前(から拡張子を除いた部分)をコマンドラインから入力すると以下の手順で実行に移されます．

①外部コマンドを100H番地以降にロードする
②100H番地をコールする

● 外部コマンドの作成

100H番地に置かれて動作するようなマシン語ルーチンを作成し，それに“COM”という拡張子を持つファイル名を付けてセーブすれば，そのままMSX-DOSから実行可能な外部コマンドとなります．

たとえば，1文字出力ルーチン(システムコール参照)を使ってコントロールコード“0CH”を出力し，画面のクリアを行うプログラムを考えます．これは次のように8バイトのサイズになります．

リスト3.1 CLS.COMの内容

```
1E 0C           LD      E,0CH    ; E := control-code of CLS
0E 02           LD      C,02H    ; C := function No. of CONSOLE OUTPUT
CD 05 00        CALL    0005H    ; call BDOS
C9              RET
```

この8バイトをCLS.COMという名前のファイルに書き込めば，画面をクリアする外部コマンド“CLS”が完成します．次のサンプルプログラムは，BASICのシーケンシャルファイル・アクセスの機能を利用し，これを作るものです．プログラムを実行し終えると，CLSコマンドがディスケット上に作成されています．MSX-DOSに戻って動作を確かめてください．

リスト3.2　CLS.COMの作成

```
100 '===== This program makes "CLS.COM" =====
110 '
120 OPEN "CLS.COM" FOR OUTPUT AS #1
130 '
140 FOR I=1 TO 8
150   READ D$
160   PRINT #1,CHR$(VAL("&H"+D$));
170 NEXT
180 '
190 DATA 1E,0C,0E,02,CD,05,00,C9
```

● 外部コマンドへの引数の受け渡し

引数を持った外部コマンドを作成する場合，コマンドラインからその外部コマンドに引数を渡す方法が2つあります．まず，ファイル名を引数としてコマンドに渡したい時には，システム・スクラッチエリアの5CH番地と6CH番地を用います．COMMAND.COMは外部コマンド実行の際，つねにコマンドラインの第1パラメータおよび第2パラメータをファイル名とみなし，それをドライブ番号(1バイト)＋ファイル名前部(8バイト)＋拡張子(3バイト)に展開して5CH番地および6CH番地に格納します．これはFCBの先頭の12バイトと同じ形式を持っていますので，これらの番地をFCBの先頭アドレスとして設定することによって，ファイルのオープンをはじめとする種々の操作が可能です．ただし，この方法では両FCBの先頭アドレスの差は16バイトしかありませんから，完全なFCBとして使用できるのは5CH番地または6CH番地の片方だけです．次に，数値などのファイル名以外の引数をコマンドに渡したい場合，または3つ以上のファイル名を扱うような外部コマンドを作成する場合には，DMA領域(初期設定アドレス0080H)を用います．DMA領域には，COMMAND.COMによって，その外部コマンドが起動されたコマンドライン全体からコマンド名自身を除いた部分が，そのバイト数(1バイト)＋コマンドライン本体の形式で格納されていますから，外部コマンド内でそれを適当に解釈して利用することが可能です．4章のリスト3.3にこのDMA領域を用いた引数の受け渡しの例がありますので参照してください．

3章 ディスクファイルの構造

システムコールを用いてディスクをアクセスする場合にどうしても必要なのは，ディスク上のデータがどのような構造を持ち，どんな方法で管理されているか，という情報です．そこで本章では，まずMSX-DOSにおけるディスクとのデータのやりとりの基本単位となる“論理セクタ”の説明から始め，最終的には人間にとって最も便利な“ファイル”によるデータ管理の方法まで話を進めます．

3.1 ディスク上のデータ単位

● セクタ

MSX-DOSは，3.5インチ2DDでもハードディスクでも，いかなるタイプのディスクドライブも基本的にはアクセスが可能です．それらの異なるメディアを統一的に扱うため，システムコールでは“論理セクタ”をディスク上のデータの基本単位として考えます． 論理セクタは0から順に付けられた番号によって指定されます．

● クラスタ

システムコールを使用している限りでは，前述のとおりセクタがデータの基本単位であると考えて間違いはありません．ところが，実際にはディスク上のデータは複数のセクタから成る“クラスタ”を単位として管理されています． FATの項で述べるように，クラスタは2から始まる通し番号で指定され，データ領域の先頭がクラスタ#2の位置に相当します．1クラスタが何セクタから成るかという情報を得るには，ファンクション1BH（ディスク情報の獲得）のシステムコールを用います．

● クラスタからセクタへの換算

後述のディレクトリやFCBには，ディスク上のデータの位置がクラスタで示されている部分があります．クラスタで示されたこれらのデータをシステムコールでアクセスするためには，あるクラスタが何番のセクタに対応しているか，という関係を求めなければなりません．これは，クラスタ#2とデータ領域の開始セクタが同一の位置に存在しているという事実をもとに，以下の

ように計算することができます．

1．与えられたクラスタ番号をCとする．
2．データ領域の開始セクタを調べ(DPBを読む)，これをS0とする．
3．1クラスタが何セクタに相当するか調べ(ファンクション1BH)，これをnとする．
4．求めるセクタ番号SはS＝S0＋(C－2)＊nの計算で得られる．

MSX-DOSでは，ディスクの中のセクタを**表3.9**に示す4つの領域に分けています．ディスクに書き込まれたファイルデータの本体は“データ領域”の部分に記録され，それ以外の3つの領域にはデータを管理するための情報が書き込まれています．これらの位置関係は**図3.9**のとおりです．ブートセクタはかならずセクタ#0に存在しますが，FAT，ディレクトリ，データ領域の開始セクタの位置はメディアによって異なるため，DPBを参照する必要があります．

ブートセクタ	MSX-DOSの起動プログラムとディスク固有の情報
FAT	ディスク上のデータの物理的な管理情報
ディレクトリ	ディスク上のファイルの管理情報
データ領域	実際のファイルデータ

表3.9　ディスクの内容

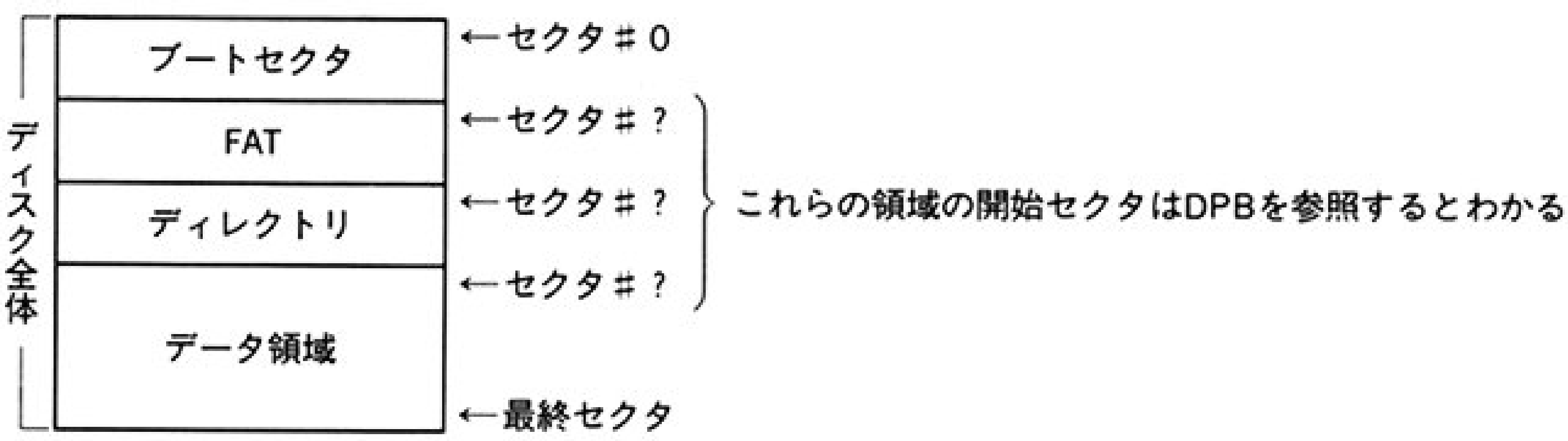


図3.9　ディスクの中の要素の位置関係

● DPB(ドライブ・パラメータ・ブロック)とブートセクタ

MSX-DOSでは，接続されている個々のドライブごとに“DPB”という領域がメモリ上のワークエリアに設けられ，各ドライブに固有の情報が記録されます．MSX-DOSは3.5インチ2DD，5インチ2Dなど，どのようなタイプのディスクドライブにも対応可能となっていますが，それはこのDPBを参照して個々のドライブに対応した処理を行うことによって，メディア間の差異が吸収できるからに他なりません．

DPBに書き込まれる情報は，もともとはディスク上のブートセクタ(セクタ#0)に存在しているものであり，それがMSX-DOSの起動時に読み出されてきます．ただし，ブートセクタの内容とDPBの内容には**図3.10**と**図3.11**に示すように，データの並べ方などの点で多少の違いがあります．

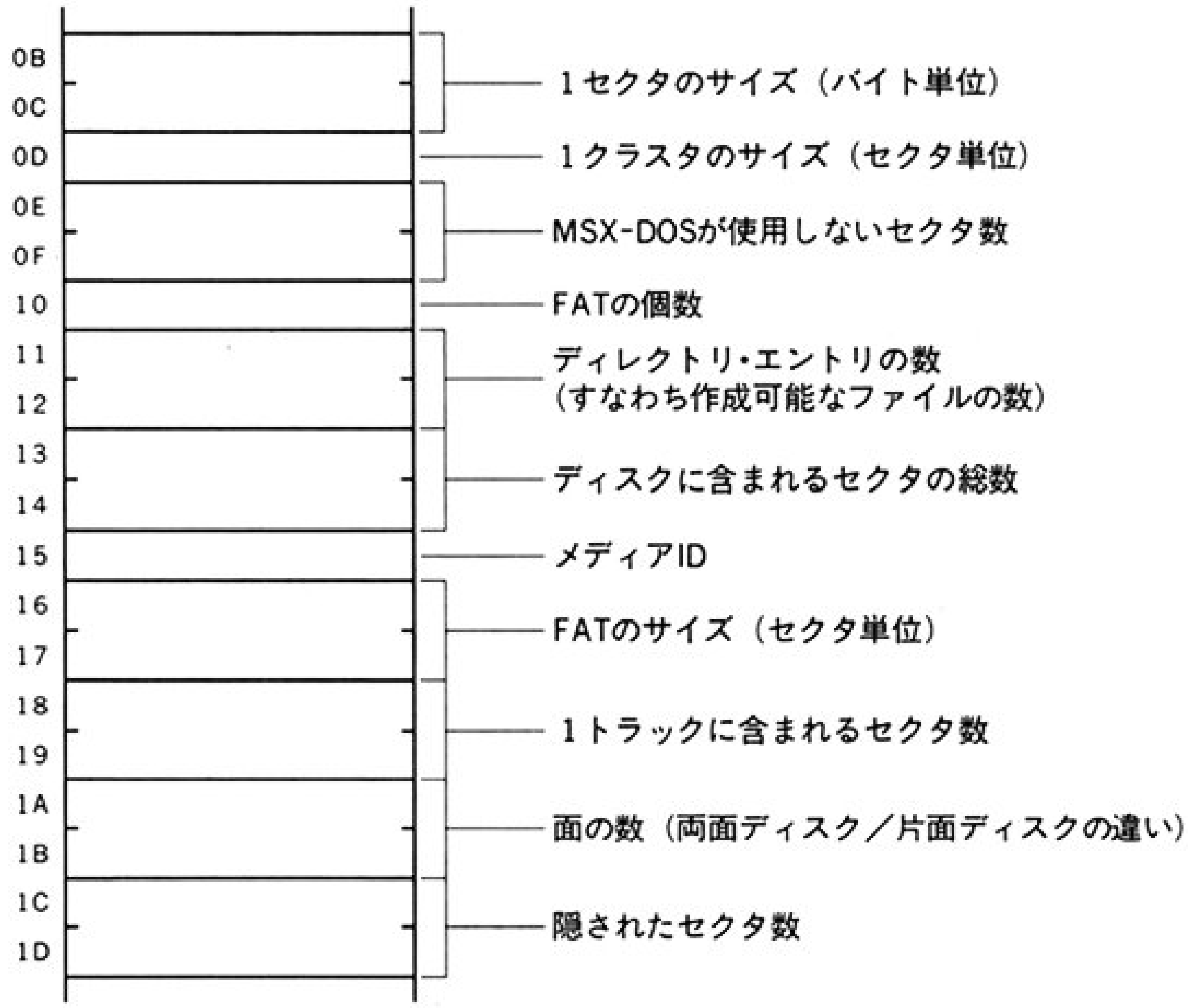


図3.10　ブートセクタの情報

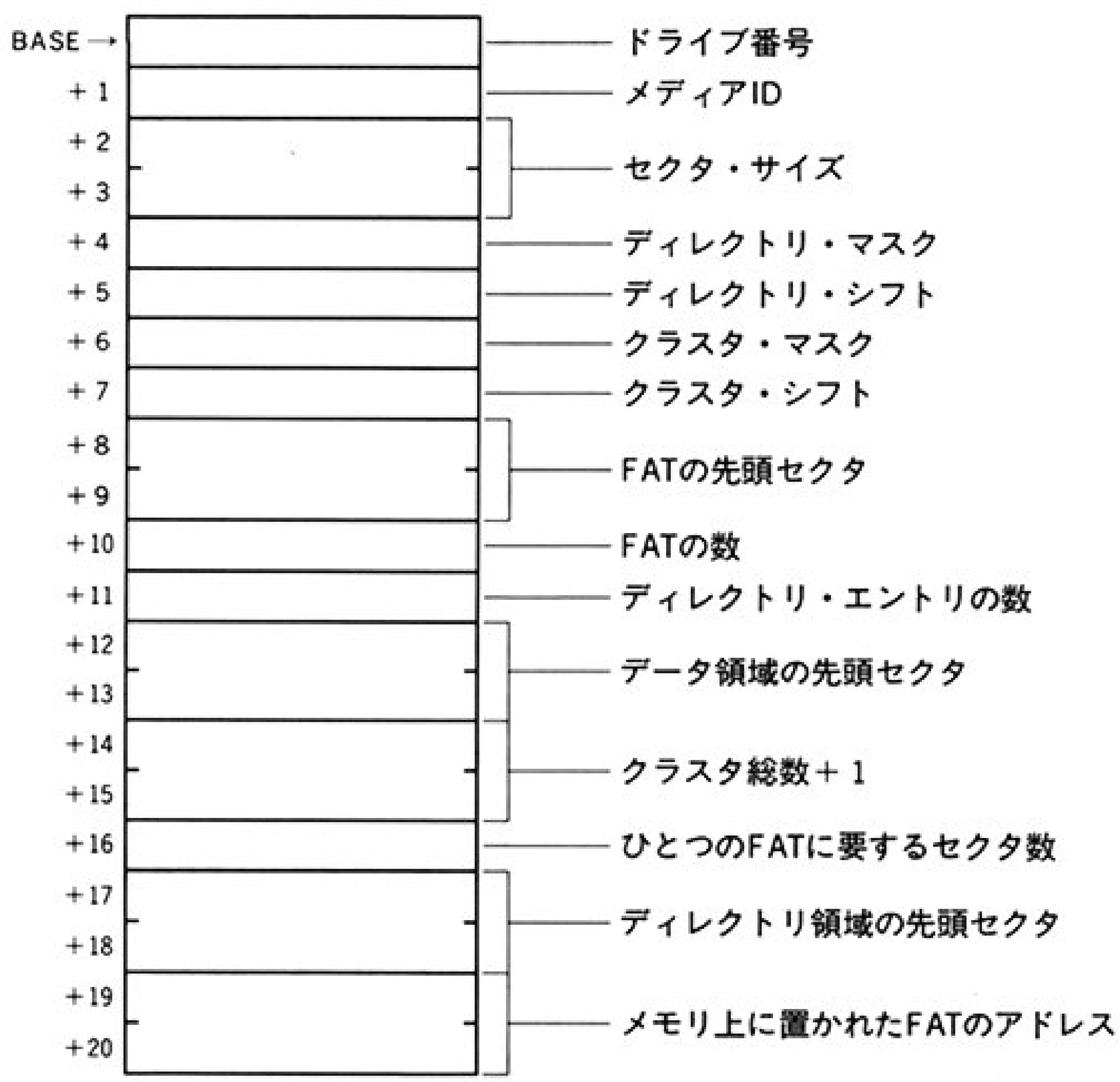


図3.11　DPBの構造

DPBをアクセスするためには，ファンクション1BH(ディスク情報の獲得)のシステムコールを用います．このシステムコールは，DPBが存在するメモリ上のアドレスとその他ブートセクタに書かれていたドライブごとの情報を返します(詳しい使用法については，4章システムコール使用法を参照)．

● FAT(ファイル・アロケーション・テーブル)

MSX-DOSは，ディスクに書き込みを行う際“クラスタ”をデータの単位とし，1クラスタよりも大きなサイズのファイルは，複数のクラスタにまたがって記録されます．しかし，その時連続したクラスタが使用されるとは限りません．特にファイルの作成／削除を何度も繰り返した後になると，使用されなくなったクラスタがディスク上のあちこちに散在します．この状態でサイズの大きなファイルを作成すると，ファイルはまったくバラバラの位置に分割して置かれることになり，何番目のクラスタは何番目のクラスタに続いている，というリンク情報を覚えておく場所が必要になります．それがFATの役割です．また，不良クラスタが発見された場合，以後そこをアクセスしないように位置を記録する目的にもFATが利用されます．FATに記録されるこのようなクラスタのリンク情報や不良クラスタ情報は，ディスクファイルを管理する上で不可欠なものであり，一部でも破損してしまうとディスク全体が使用できなくなる恐れがあります．そのためFATはつねに複数個が用意され，万一の場合に備えています．

FATの例を図3.12に示します．先頭の1バイトは“FAT ID”と呼ばれ，ディスクのメディアタイプを示す値(表3.2のメディアIDと同じ値)，次の2バイトは意味のないダミー値が入り

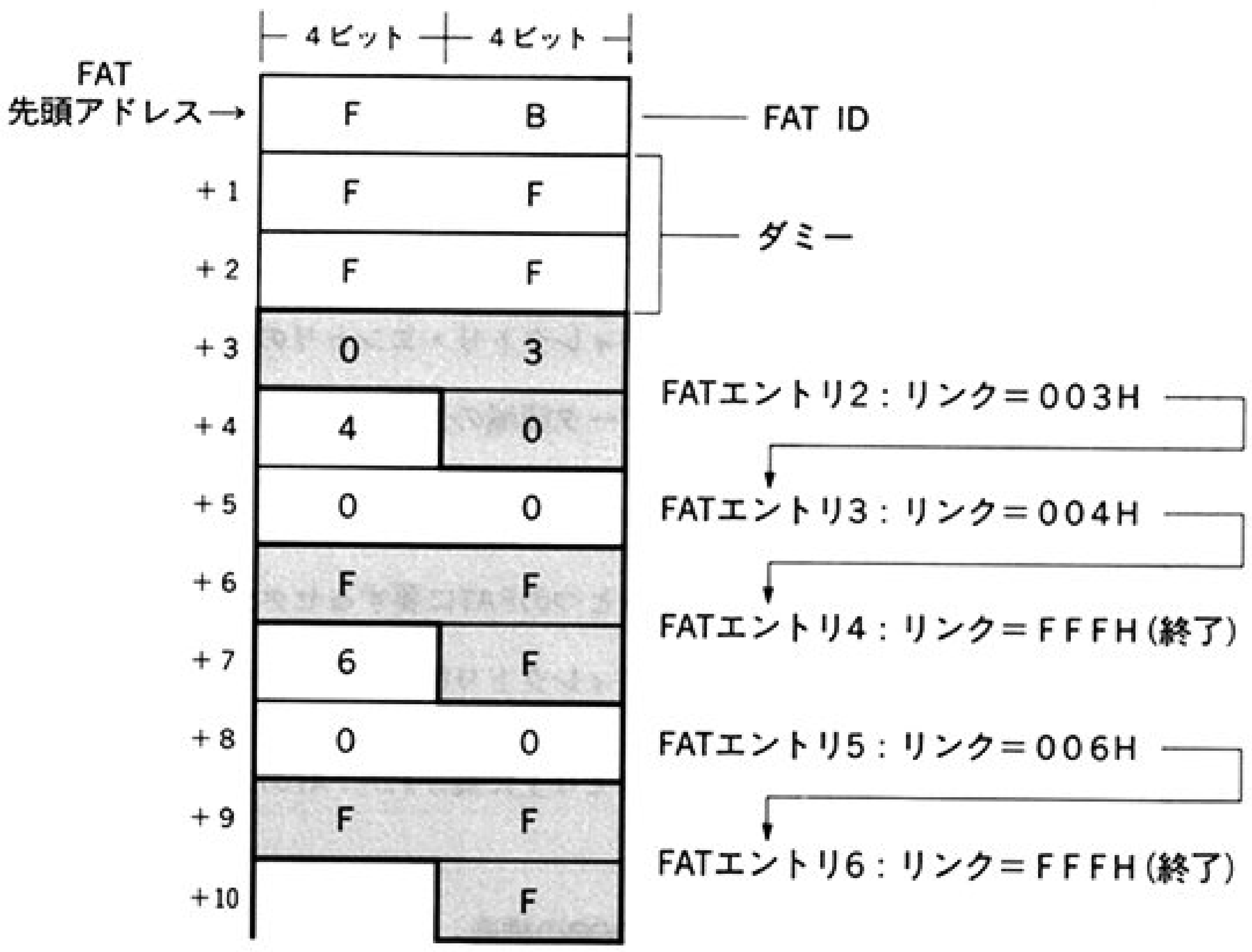


図3.12　FATの実例

ます．そしてその次から，１クラスタにつき12ビットという変則的なフォーマットで実際のリンク情報が記録されます．それぞれのリンク情報を記録している12ビットの領域はFATエントリと呼ばれます．FATエントリは２番から始まることに注意してください．FATエントリの番号は，それに対応するクラスタの番号でもあります．FATエントリに記録された12ビットのリンク情報は，図３.13の要領で読み取ってください．

リンク情報は，次に続くクラスタ番号を示す値です．もしFFFHとなっている場合は，そのクラスタでファイルが終了したことを意味します．図３.12の例では，クラスタ#２→クラスタ#３→クラスタ#４，という３クラスタ分の大きさのファイルと，クラスタ#５→クラスタ#６，の２クラスタ分のファイルが存在していることがわかります(クラスタが番号の小さい順にリンクしているのは図を見やすくするためで，実際には番号順である必要はない)．

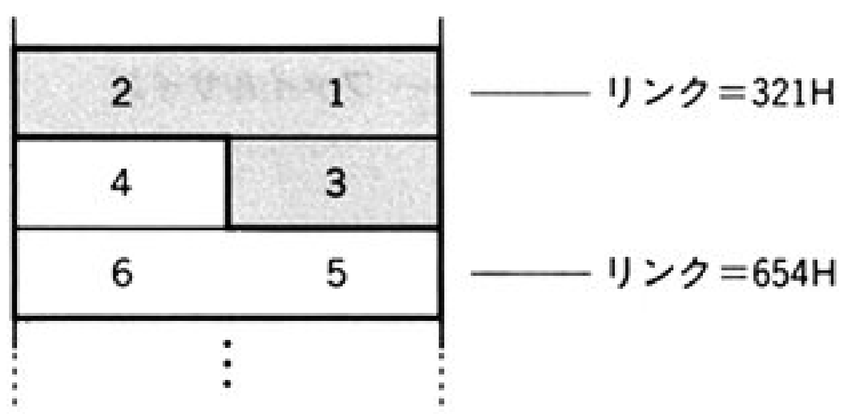


図3.13　FATの読み方

● ディレクトリ

前述のFATはディスク上のデータの物理的な位置関係を表すものであり，そこに書き込まれたデータ内容に関する情報は含んでいません．したがって，あるファイルがどのようなデータから構成されているかを知るには，FATとは別の情報源が必要です．これが“ディレクトリ”です．ディレクトリは32バイトで構成され，図３.14に示すようにファイル名，ファイル属性，作成の日付，作成の時刻，ファイルの先頭クラスタ番号，ファイルサイズの各情報を記録しています．

ディレクトリ中にある“ファイル属性”は，ファイルに不可視属性を設定するもので．このバイトの下から２ビット目を“１”にすると，そのディレクトリで指定されるファイルはシステムコールではアクセス不可能となります(図３.15参照)．ファイル属性バイトはMS-DOSのディレクトリにも同じものが存在し，そちらでは１バイト中の他のビットを用いて書き込み禁止属性などを設定することもできるのですが，MSX-DOSにはその機能は備わっていません．

日付と時刻は，図３.16，３.17に示すように，それぞれ２バイトの領域を３つのビットフィールドに分割して記録しています．“秒”用のビットフィールドが５ビットしかないため，時間の最小単位は２秒となっています．“年”は７ビットに０～99の値を設定することで，西暦1980年～2079年を表します．

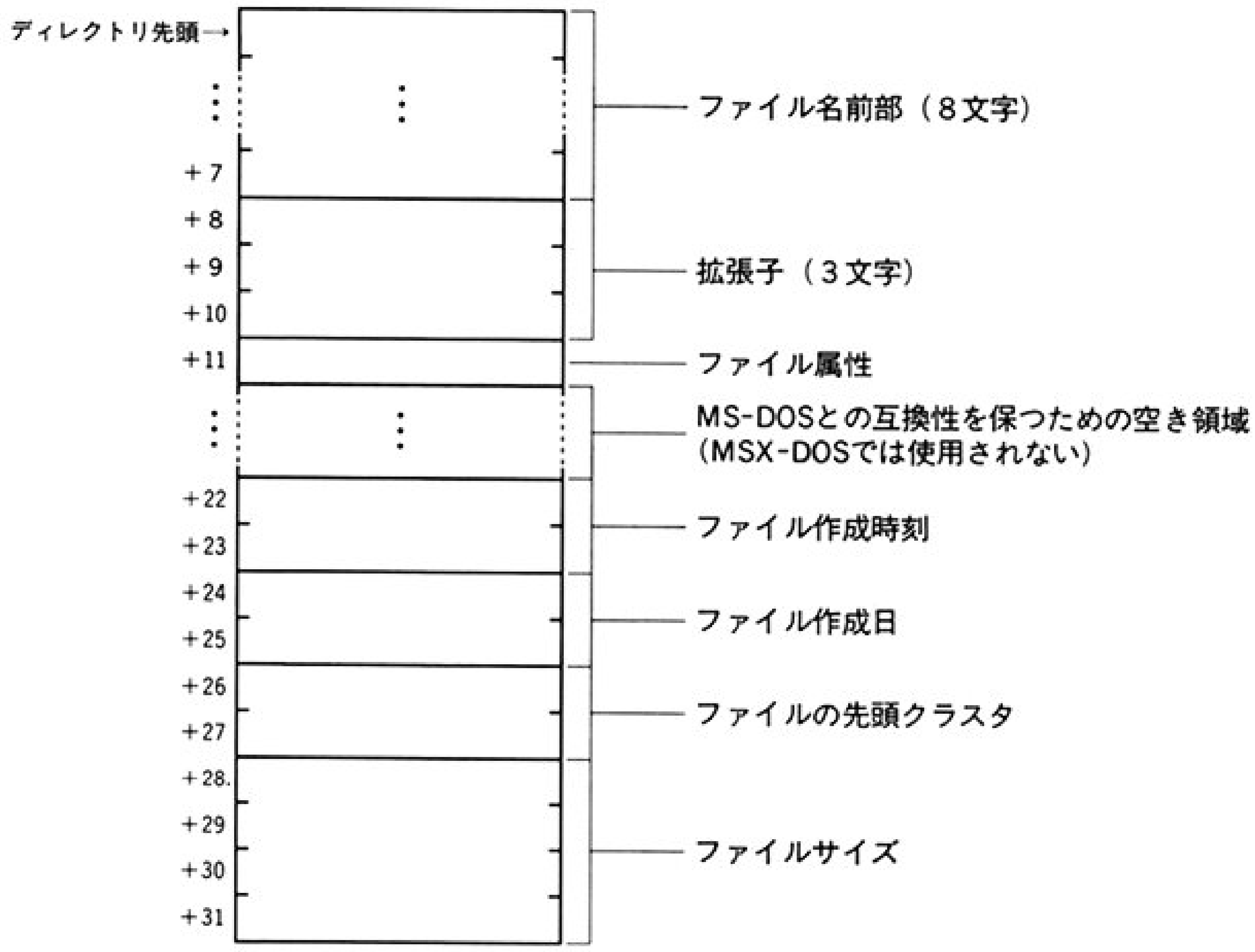


図3.14　ディレクトリの構成

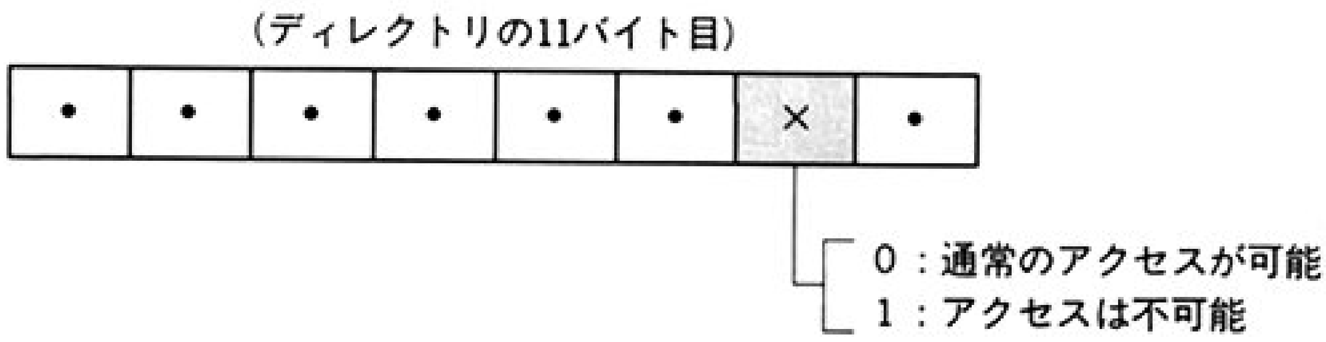


図3.15　ファイルの不可視属性

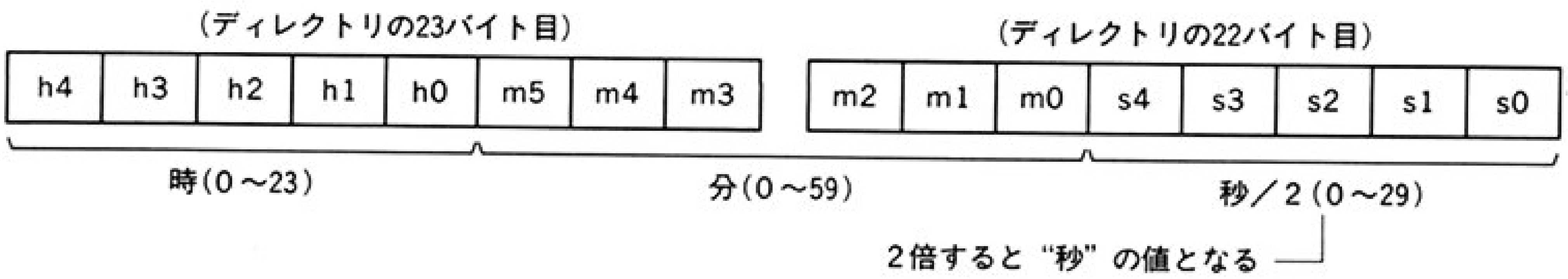


図3.16　時刻を表すビットフィールド

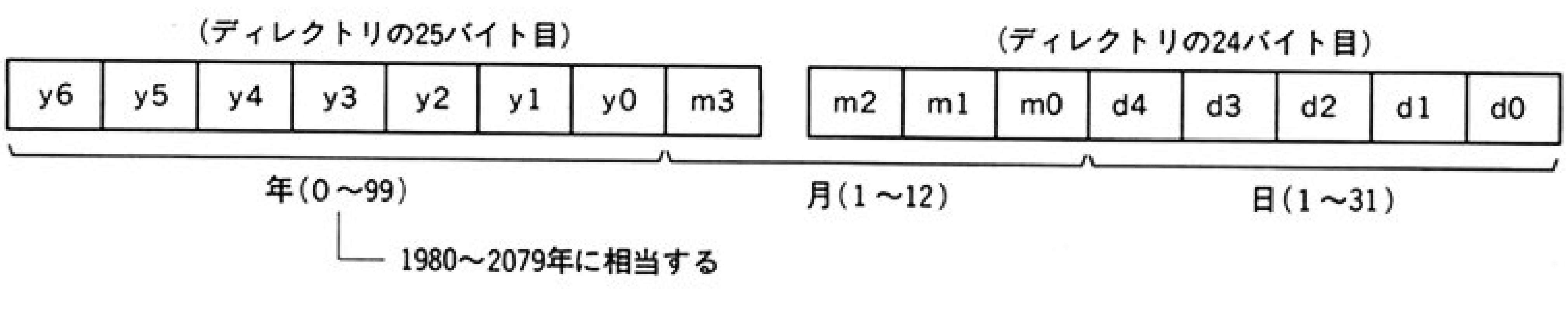


図3.17　日付を表すビットフィールド

このディレクトリ情報が実際に記録されている場所が，ディスク上のディレクトリ領域です(図3.9参照)．その位置(先頭セクタ)は，DPBに記録されています．ディレクトリ領域には，図3.8に示すように32バイトごとにディレクトリ・エントリ(ディレクトリの格納場所)が並んでいます．ファイルの作成を行うと，使われていないディレクトリ・エントリの中で，番号がいちばん小さいところに目的のファイルのディレクトリが作られます．ファイルが削除されると，該当するディレクトリ・エントリの最初の1バイトにE5Hが書き込まれ，そのディレクトリ・エントリが空いたことを示します．ディレクトリ・エントリがすべて使用されてしまうと，データ領域がいくら残っていても新しいファイルを作ることはできません．ディレクトリ・エントリの数，すなわち1枚のディスク上に作成できるファイルの数も，DPBに記録されています．

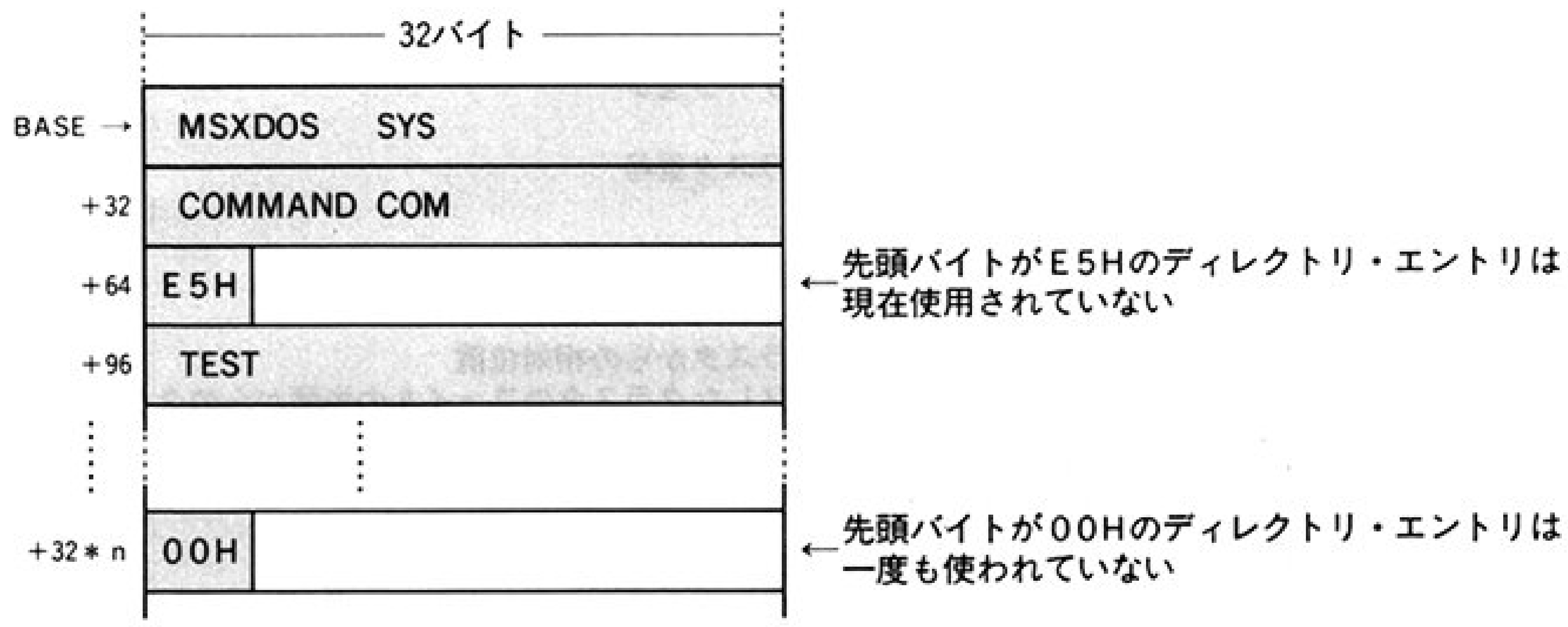


図3.18　ディレクトリ領域の構成

3.2 ファイルのアクセス

● FCB(ファイル・コントロール・ブロック)

ディレクトリ領域に記録された情報を用いると，データを“ファイル”として扱うことが可能となります．この方法の特徴は，データの位置をセクタ番号やクラスタ番号のような具体的数値で表現するのではなく，“名前”を用いてファイルを指定できるという点にあります．目的のファイルがディスク上のどのアドレスに存在しているか，ということはすべてシステムコールにまかせ，人間はただファイル名を指示するだけでそれをアクセスできるのです．この時，ディレクトリとともに大きな役割を果たすものがFCBです．

FCBは，システムコールを用いてファイルを扱う際，必要となる情報を格納しておく領域です．ひとつのファイルを扱うごとに図3.19で示すような37バイトのメモリが必要になります．FCBはメモリ上のどこに置かれていてもかまいませんが，MSX-DOSの機能を活かすため，005CHのアドレスがしばしば用いられます．

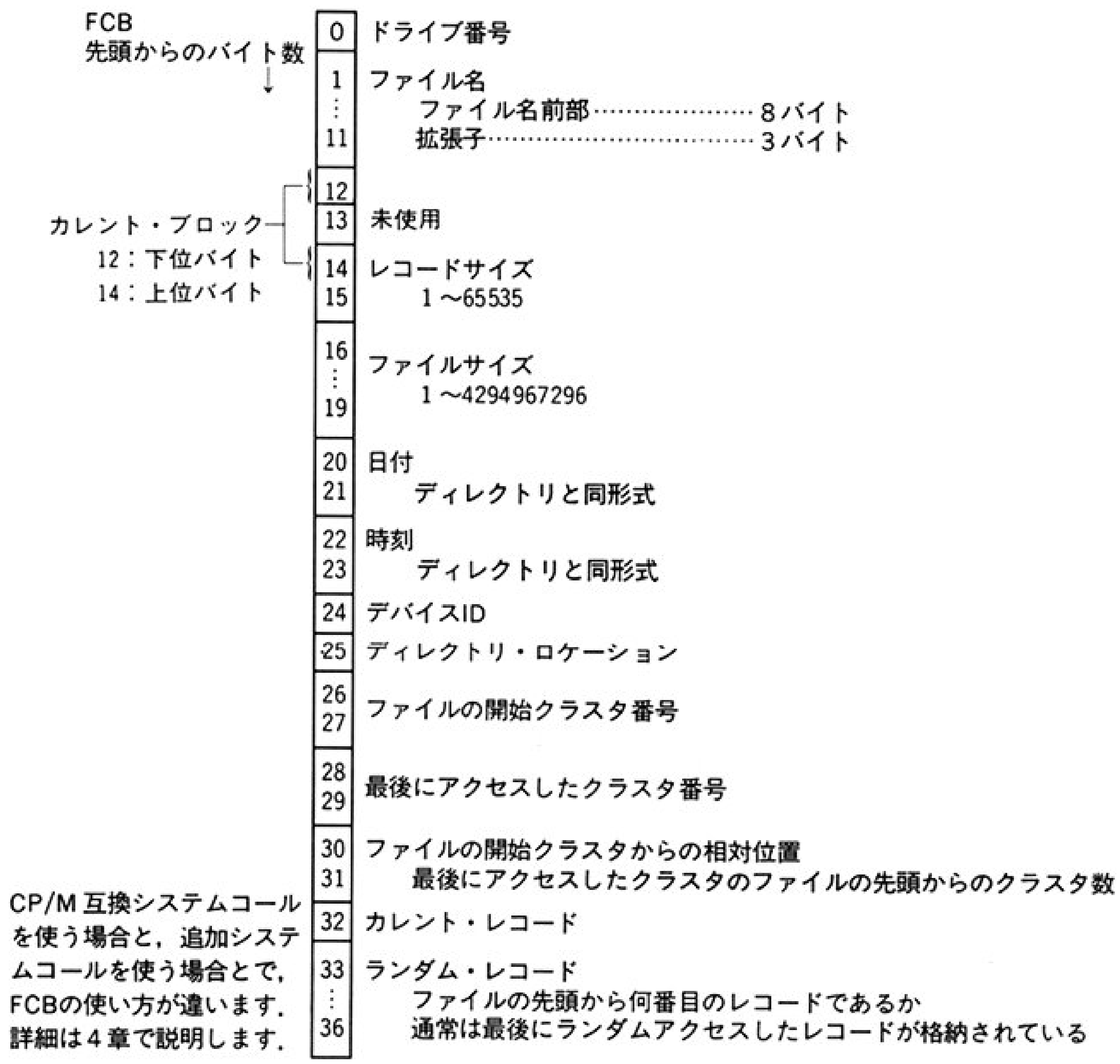


図3.19　FCBの構造

・ドライブ番号(00H)

ファイルの存在するディスクドライブを示します．
(0→デフォルト・ドライブ，1→A：　，2→B：　……)

・ファイル名前部(01H～08H)

ファイル名前部は最大8文字まで指定可能です．8文字に満たない場合は，その分がスペース(20H)で埋められます．

・拡張子(09H～0BH)

拡張子は最大3文字まで指定可能です．3文字に満たない場合は，その分がスペース(20H)で埋められます．

・カレントブロック(0CH, 0EH)

シーケンシャル・アクセスの際，参照中のブロック番号を示します(4章のファンクション14H，15H，24H参照)．0CHが下位バイト，0EHが上位バイトを表します．

・レコードサイズ(0EH～0FH)

1回のアクセスで読み出しあるいは書き込みを行うデータ単位(レコード)のサイズをバイト数で指定します(ファンクション26H，27H参照).

・ファイルサイズ(10H～13H)

バイト単位で，ファイルの大きさを示します．

・日付(14H～15H)

最後にファイルに書き込みを行った日付を示します．フォーマットはディレクトリに記録されているものと同様です．

・時刻(16H～17H)

最後にファイルに書き込みを行った時刻を示します．フォーマットはディレクトリに記録されているものと同様です．

・デバイスID(18H)

周辺装置をファイルとしてオープンする場合，このデバイスIDフィールドに**表3.10**のような値が設定されます．通常のディスクファイルの場合には，このフィールドの値は40H＋ドライブ番号です．

デバイス名	デバイスID
CON (Consore)	0FFH
PRN (Printer)	0FBH
LST (List=Printer)	0FCH
AUX (Auxiliary)	0FEH
NUL (Null)	0FDH

表3.10　デバイスID

・ディレクトリ・ロケーション(19H)

ディレクトリ領域の中で，何番目のディレクトリ・エントリに該当するファイルであるかを示します．

・先頭クラスタ(1AH～1BH)

ディスクにおけるファイルの先頭のクラスタを示します．

・最終アクセスクラスタ(1CH～1DH)

最後にアクセスされたクラスタを示します．

・最終アクセスクラスタの先頭クラスタからの相対位置(1EH～1FH)

最後にアクセスされたクラスタの先頭クラスタからの相対位置を示します．

・カレントレコード(20H)

シーケンシャルアクセスの際，現在参照中のレコードを示します(ファンクション14H，15H，24H参照)．

・ランダムレコード(21H〜24H)

ランダムアクセスおよびランダムブロックアクセスの際，アクセスしたいレコードを指定します．前記のレコードサイズフィールドを1〜63の値に設定していると21H〜24Hの4バイト全体が使用されますが，レコードサイズが64以上の場合には21H〜23Hの3バイトしか意味を持ちません(ファンクション26H，27H参照)．

● ファイルのオープン

FCBを用いてファイルをアクセスするには，まずファイルをオープンする手続きが必要になります．“ファイルのオープン”とは，システムコールのレベルでは，ファイル名フィールドだけが定義された不完全なFCBを，ディレクトリエリアに記された情報を用いて完全なFCBに変換することを意味しています．図3.20に“オープンされていないFCB”と“オープンされたFCB”の違いを示します．

● ファイルのクローズ

ファイルをオープンして書き込みを行った場合，それに伴って，ファイルサイズをはじめとするFCBの各フィールドの内容も変更を受けます．この更新されたFCBの情報をディレクトリ領域に戻しておかないと，次回ファイルをアクセスする際に，ディレクトリの情報と実際のファイルの内容がくい違ってしまいます．更新されたFCBの情報をディレクトリに戻すというこの操作が，システムコールにおけるファイルのクローズに相当します．

● ランダム・ブロック・アクセス(レコードによるファイル管理)

MSX-DOSには，“RANDOM BLOCK READ”および“RANDOM BLOCK WRITE”という，たいへん有能なシステムコールが存在します．このシステムコールを用いると，ファイルを任意の大きさのデータ単位に分割し，その先頭から順に0, 1, 2…という番号を付けて管理できます．このデータ単位を“レコード”と呼びます．レコードのサイズは1バイト以上であればどんな値を設定してもかまいません．ひとつのファイル全体を1レコードとして扱うこと(究極のシーケンシャルアクセス)も，データ1バイトずつを1レコードとして扱うこと(究極のランダムアクセス)も，128バイトを1レコードとして扱うこと(CP/M方式)も可能です．

その際，FCBの“レコードサイズ”と“ランダムレコード”の両フィールドがレコードの指定に用いられます．レコードサイズ・フィールドの値は1レコードのバイト数を示します．ランダ

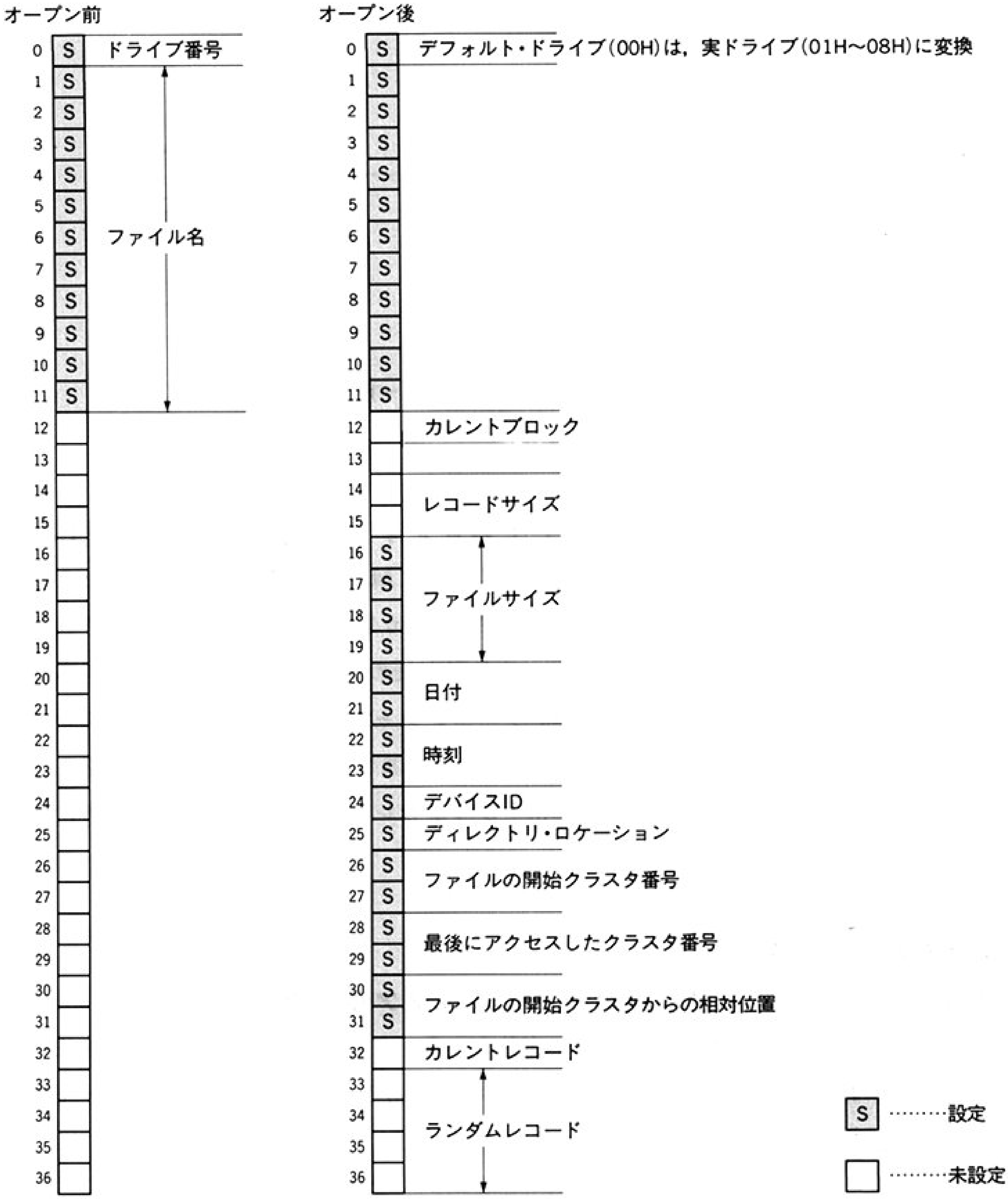


図3.20　FCBのオープン前後

ムレコード・フィールドは，アクセスするレコードの番号を任意に指定できます(詳しい使用法については，それぞれのシステムコールの説明を参照)．

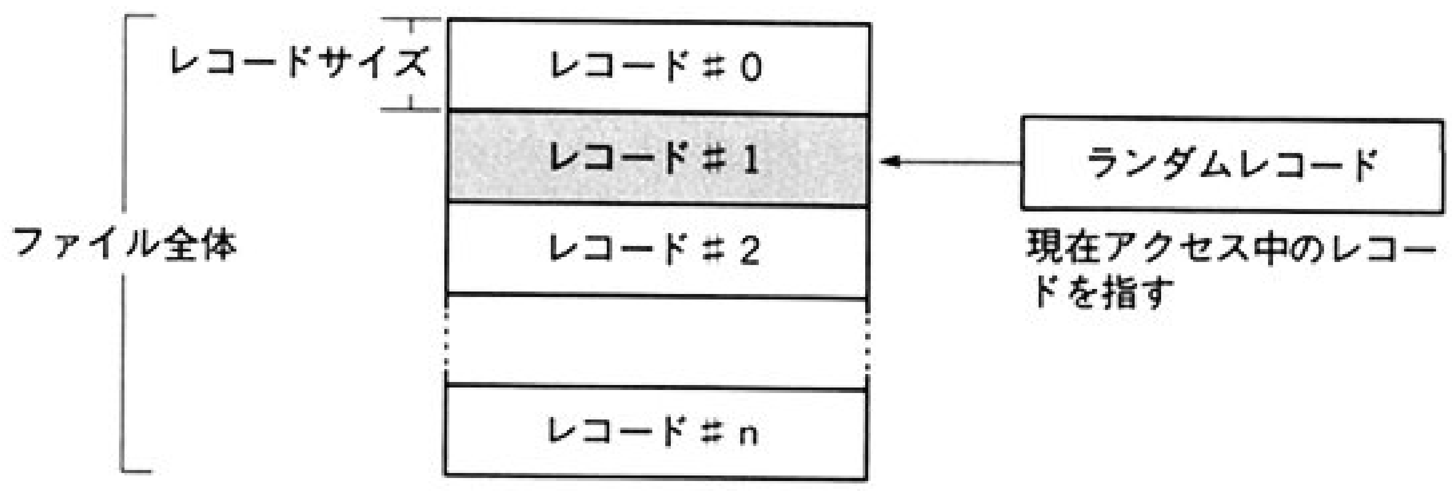


図3.21　ファイルとレコード

● シーケンシャルアクセス
(固定長レコード＋カレントレコード＋カレントブロックによるファイル管理)

CP／Mとの互換性を保つため，MSX-DOSではCP／Mと同様のファイルアクセスをすることも可能です．そのひとつが，“カレントレコード”と“カレントブロック”で管理されるシーケンシャルアクセスです．この方法では，128バイト固定長のレコードをデータの基本単位とします．ファイルのアクセスはつねに先頭から順(シーケンシャル)に行い，アクセスしたレコード数はFCBのカレントレコード・フィールドでカウントされます．カレントレコード・フィールドの値は128に達すると0にリセットされ，その桁上がりがカレントブロック・フィールドにカウントされます．

● ランダムアクセス
(固定長レコード＋ランダムレコードによるファイル管理)

CP／Mとの互換性を保つための2つ目の方法が，ランダムレコード・フィールドを用いたランダムアクセスです．任意の位置のレコードをアクセスすることが可能ですが，レコードサイズは128バイト固定長です．

4章 システムコールの使用法

システムコールとは，MSX-DOSの基本的な入出力操作を行うBDOSを，汎用のサブルーチンとしてまとめたもので，あらかじめ決められた手順にしたがってシステムコールを呼び出すことによってMSXのディスクシステムが持つ基本的な機能が簡単に実行できます．

システムコールの役割には，大きく分けると次の2つがあります．1つは，基本的な機能をあらかじめ用意することによってユーザーの負担を少なくすること．もう1つは，すべてのプログラムが基本的な機能を共有することによって，移植性や汎用性を高めることです．システムコールを自由自在に活用することによって，プログラム開発の期間は短縮され，できあがったプログラムは移植性の高いものとなります．

システムコールを実行するには，Z80CPUのCレジスタに決められたファンクション番号を入れ，次のアドレスをコールします．

0005H ································ MSX-DOS
F37DH(&HF37D) ··············· MSX DISK-BASIC

たとえば，ファンクション番号が01FHであり，前準備としてAレジスタに00Hをセットすることになっているシステムコールがあるとすると，MSX-DOSの場合，このシステムコールは次のようにして呼び出します．

```
LD    A, 00H
LD    C, 01FH
CALL  0005H
  :
```

CALL文の後には，戻り値(処理の結果)を受け取ったり，退避していたレジスタを復帰させたりする事後処理が続きます．システムコールはDISK-BASICから利用することもでき，その場合はF37DH番地がエントリ・アドレスとなります．この場合には，実際はCLEAR文で確保した領域にマシンコードを格納し，その先頭アドレスをUSR関数で呼び出してやることになります．

● システムコールの形式

本章では，以下の書式によってシステムコールの使用法を紹介します．

ファンクション：	ファンクション番号
設定：	システムコールを行う前にレジスタやメモリ上にセットすべき値
戻り値：	システムコールから戻った時にセットされている値

ファンクション：

ファンクション番号とは，さまざまなシステムコールの種類を判別するためのもので，それぞれのシステムコールに対応する番号が決められています．システムコールを呼び出す時には，このファンクション番号をCレジスタにセットします．

設定：

また，システムコールの前準備として，レジスタやメモリに必要な値をセットしなければならないことがありますが，これを本章では“設定：”の部分に示します．

戻り値：

システムコールの結果得られた値は，多くの場合レジスタやメモリの内容にセットされることになります．これを，本章では出力と呼びますが，この出力がどこに，どのようにセットされているかを“戻り値：”の部分に示します．

ただし，システムコールを使用した後では，この戻り値としてセットされるレジスタやメモリ以外のレジスタの内容が破壊されていることがあります．そこで，システムコールを使用する場合には，そのシステムコールを呼び出す前に，破壊されては困るレジスタの内容を，適当な場所(スタックなど)に退避させておくようにしてください．

MSXのシステムコールは**表3.11**に示す42個ありますが，本章ではこれらを次の4つに分類して説明します．

・周辺装置とのI/O
・環境設定
・アブソリュートREAD/WRITE(セクタの直接アクセス)
・FCBを用いたファイルアクセス

ファンクションNo.	機　　能
00H	システムリセット
01H	コンソールから1文字入力(入力待ちあり，エコーバックあり，コントロールコードチェックあり)
02H	コンソールへ1文字出力
03H	補助入力装置から1文字入力
04H	補助出力装置へ1文字出力
05H	プリンタへ1文字出力
06H	コンソールから1文字入力(入力待ちなし，エコーバックなし，コントロールコードチェックなし)／1文字出力
07H	コンソールから1文字入力(入力待ちあり，エコーバックなし，コントロールコードチェックなし)
08H	コンソールから1文字入力(入力待ちあり，エコーバックなし，コントロールコードチェックあり)
09H	文字列出力
0AH	文字列入力
0BH	コンソールからの入力チェック
0CH	バージョン番号の獲得
0DH	ディスクリセット
0EH	デフォルト・ドライブの選択
0FH	ファイルのオープン
10H	ファイルのクローズ
11H	ワイルドカードに一致する最初のファイルの検索
12H	ワイルドカードに一致する2番目以降のファイルの検索
13H	ファイルの抹消
14H	シーケンシャルなファイルの読み出し
15H	シーケンシャルなファイルの書き込み
16H	ファイルの作成
17H	ファイル名の変更
18H	ログイン・ベクトルの獲得
19H	デフォルト・ドライブ名の獲得
1AH	DMAアドレスの設定
1BH	ディスク情報の獲得
1CH～20H	無効
21H	ランダムなファイルの読み出し
22H	ランダムなファイルの書き込み
23H	ファイルサイズの獲得
24H	ランダムレコード・フィールドの設定
25H	無効
26H	ランダムブロック書き込み
27H	ランダムブロック読み出し
28H	ランダムなファイルの書き込み（不用部を00Hで埋める）
29H	無効
2AH	日付の獲得
2BH	日付の設定
2CH	時刻の獲得
2DH	時刻の設定
2EH	ベリファイ・フラグの設定
2FH	論理セクタの読み出し
30H	論理セクタの書き込み

表3.11　システムコール一覧

● 使用上の注意

システムコールのファンクション番号は00Hから30Hまでですが，この中には以下に示す7個の無効なシステムコールがあります．

1CH～20H, 25H, 29H

これらの無効なシステムコールを呼び出した場合には，Aレジスタに00Hがセットされる以外は何も行いません．また，ファンクション31H以降のシステムコールは未定義であり，使用した場合の結果は不定(安全性を保証できない)となりますので注意してください．

リスト3.3 ユーティリティ・ルーチン

```
;==============================================================
;
;  List 3.3   utility.mac
;
;    these routines are used in other programs
;
;    GETARG, STOHEX, PUTHEX, PUTCHR, DUMP8B
;
;==============================================================
;
        PUBLIC  GETARG
        PUBLIC  STOHEX
        PUBLIC  PUTHEX
        PUBLIC  PUTCHR
        PUBLIC  DUMP8B

BDOS    EQU     0005H
DMA     EQU     0080H

;----- DE := address of arg(A)'s copy -----

GETARG: PUSH    AF
        PUSH    BC
        PUSH    HL

        LD      C,A
        LD      HL,DMA
        LD      B,(HL)
        INC     HL
        INC     B

SKPARG: DEC     B
        JR      Z,NOARG
SKP1:   LD      A,(HL)
        INC     HL
        CALL    TRMCHK
        JR      NZ,SKP1
SKP2:   LD      A,(HL)
        INC     HL
        CALL    TRMCHK
        JR      Z,SKP2
        DEC     HL
        DEC     C
        JR      NZ,SKPARG

CPYARG: LD      DE,BUFMEM
CPY1:   LD      A,(HL)
        LD      (DE),A
        INC     HL
        INC     DE
        CALL    TRMCHK
        JR      NZ,CPY1

        DEC     DE
        LD      A,'$'
        LD      (DE),A
        LD      DE,BUFMEM
        JR      EXIT

NOARG:  LD      DE,BUFMEM
        LD      A,'$'
        LD      (DE),A
```

本プログラムリストに含まれる5つのユーティリティ・ルーチンは，後述のサンプルプログラム中で利用される．

デフォルトのDMA領域(0080H～)に格納されたコマンドラインの第nパラメータ（nはAレジスタで指定）をメモリに取り込み，その先頭アドレスをDEレジスタに返す．

```
EXIT:   POP     HL
        POP     BC
        POP     AF
        RET

TRMCHK: CP      09H
        RET     Z
        CP      0DH
        RET     Z
        CP      ' '
        RET     Z
        CP      ':'
        RET

;----- HL := hexadecimal value of [DE] -----
```

DEレジスタで指される16進文字列を2バイト整数に変換し，HLレジスタに格納する．

```
STOHEX: PUSH    AF
        PUSH    DE
        LD      HL,0000H
        CALL    STOH1
        POP     DE
        POP     AF
        RET

STOH1:  LD      A,(DE)
        INC     DE
        SUB     '0'
        RET     C
        CP      10
        JR      C,STOH2
        SUB     'A'-'0'
        RET     C
        CP      6
        RET     NC
        ADD     A,10
STOH2:  ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        OR      L
        LD      L,A
        JR      STOH1

;----- print A-reg. in hexadecimal form (00-FF) -----
```

Aレジスタの内容を16進2桁で表示する．

```
PUTHEX: PUSH    AF
        RR      A
        RR      A
        RR      A
        RR      A
        CALL    PUTHX1
        POP     AF
PUTHX1: PUSH    AF
        AND     0FH
        CP      10
        JR      C,PUTHX2
        ADD     A,'A'-10-'0'
PUTHX2: ADD     A,'0'
        CALL    PUTCHR
        POP     AF
        RET
```

```
;----- put character -----

PUTCHR: PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        LD      E,A
        LD      C,02H
        CALL    BDOS
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        RET

;----- dumps 8byte of [HL]~[HL+7] in hexa & ascii form -----

DUMP8B: PUSH    HL
        LD      B,8
DUMP1:  LD      A,(HL)
        INC     HL
        CALL    PUTHEX
        LD      A,' '
        CALL    PUTCHR
        DJNZ    DUMP1
        POP     HL
        LD      B,8
DUMP2:  LD      A,(HL)
        INC     HL
        CP      20H
        JR      C,DUMP3
        CP      7FH
        JR      NZ,DUMP4
DUMP3:  LD      A,'.'
DUMP4:  CALL    PUTCHR
        DJNZ    DUMP2
        LD      A,0DH
        CALL    PUTCHR
        LD      A,0AH
        CALL    PUTCHR
        RET

;----- work area -----

BUFMEM: DS      256

        END
```

HLレジスタで示すアドレスから8バイト分のメモリの内容を，16進およびキャラクタコードでダンプする．

4.1 周辺装置とのI/O

入出力に利用するシステムコールであり，コンソール(画面／キーボード)やAUX(外部入出力)，プリンタなどに関する制御を行います．文字列入力やプリンタ出力などのサブルーチンを作成するのは，多くのプログラムに必要なルーチンワークですが，本節のシステムコールを利用することにより，簡単に汎用的なプログラムを組むことができます．

● コンソール入力

ファンクション：01H

設定：なし

戻り値：Aレジスタ←コンソールから入力した1文字

入力がない場合(キーが押されておらず，入力バッファも空の場合)には，入力待ちを行う．入力された文字はコンソールにエコーバックされる．以下に示すコントロールキャラクタの入力が可能であり，Ctrl-Cを受け付けるとプログラムの実行を中断してMSX-DOSのコマンドレベルに戻る．Ctrl-Pを受け付けると，以後の入力はすべてプリンタにもエコーバックされるようになり，この状態は，Ctrl-Nを受け付けることで解除される．

Ctrl-C　システム・リセット
Ctrl-P　プリンタへのエコー開始
Ctrl-N　プリンタへのエコー停止

● コンソール出力

ファンクション：02H

設定：Eレジスタ←出力する文字コード

戻り値：なし

Eレジスタで指定した文字を画面に表示する．Ctrl-Sの入力があると表示動作を停止するが，任意のキー押すことによって再び表示動作を開始する．

● 外部入力

ファンクション：03H

設定：なし

戻り値：Aレジスタ← AUXデバイスから読み込んだ1文字

● 外部出力

ファンクション：04H

設定：Eレジスタ←AUXデバイスに出力するキャラクタコード

戻り値：なし

● プリンタ出力

ファンクション：05H

設定：Eレジスタ←プリンタに出力するキャラクタコード
戻り値：なし

● 直接コンソール入出力

ファンクション：06H
設定：Eレジスタに0FFHをセットすれば入力，0FFH以外をセットすれば出力となる．Eレジスタに0FFH以外の値がセットされていた場合は，セットされた値をキャラクタコードとみなしてコンソールに出力する．
戻り値：Eレジスタが0FFHにセットされていた場合(入力)には，Aレジスタに入力の結果がセットされる．Aレジスタにセットされる値は，キーが押されていた場合はそのキャラクタコード，押されていなかった場合は00Hとなる．Eレジスタが0FFH以外にセットされていた場合(出力)には，戻り値はなし．

コントロールキャラクタのサポート，入力のエコーバックは行わない．

● 直接コンソール入力　その1

ファンクション：07H
設定：なし
戻り値：Aレジスタ←コンソールから入力した1文字

コントロールキャラクタのサポートは行わない．エコーバックは行わない．

● 直接コンソール入力　その2

ファンクション：08H
設定：なし
戻り値：Aレジスタ←コンソールから入力した1文字

エコーバックは行わない．コントロールキャラクタはファンクション01Hと同様に処理する．

● 文字列出力

ファンクション：09H
設定：DEレジスタ←メモリ上に用意した，コンソールに出力するべき文字列の先頭アドレス
戻り値：なし

文字列の最後には，終端記号として24H(“$”)を付加しておく必要がある．ファンクション02

H(コンソール出力)と同様にCTRL-Sを処理する.

● 文字列入力

ファンクション：0AH

設定：最大入力文字数(1～0FFH)をセットしたメモリのアドレスを,DEレジスタにセットする.

戻り値：DEレジスタに示されたアドレスに1を加えたアドレスにはコンソールから実際に入力された文字数を，DEレジスタに示されたアドレスに2を加えたアドレス以降には，コンソールから入力された文字列をセットする.

リターンキーの入力をコンソールからの入力の終端とみなす．ただし，入力文字数が指定の文字数(DEレジスタで示されるアドレスの内容＝1～255)を超える場合には，指定の入力文字数までを入力文字列とみなしてメモリにセットした後，処理を終了する．次の文字からリターンキーまでの入力は無視される．このシステムコールによる文字列入力時には，テンプレートによる編集が可能である.

● コンソールの状態チェック

ファンクション：0BH

設定：なし

戻り値：キーボードが押されていた場合には0FFHを，押されていなかった場00Hを，Aレジスタにセットする.

4.2　環境の設定と読み出し

デフォルト・ドライブの変更やシステムのさまざまなデフォルト値のセットなど，MSXシステムの環境設定を行うためのシステムコールです.

● システム・リセット

ファンクション：00H

設定：なし

戻り値：なし

MSX-DOS上からコールした場合には，0000Hに分岐することによってシステムがリセットされる．MSX DISK-BASIC上からコールした場合には，MSX DISK-BASICがウォームスター

ト(Warm Start)する．つまり，ロードされているプログラムを破壊せずにBASICのコマンドレベルに戻る．

● バージョン番号の獲得

ファンクション：0CH

設定：なし

戻り値：HLレジスタ←0022H

このシステムコールは，CP／Mにおける，さまざまなCP／Mのバージョン番号を獲得するためのものであるが，MSX-DOSの場合には，一律に0022Hがセットされる．

● ディスクリセット

ファンクション：0DH

設定：なし

戻り値：なし

変更されたがまだディスクに書き込まれていないセクタがあれば，それをディスクに書き込んだ後，デフォルト・ドライブをAドライブにセットし，DMAを0080Hにセットする．

● デフォルト・ドライブの設定

ファンクション：0EH

設定：Eレジスタ←デフォルト・ドライブ番号(A＝00H, B＝01H, …)

戻り値：なし

システムコールによるディスクアクセスは，特に指定しない限りデフォルト・ドライブ番号で示されるデフォルト・ドライブに対して行われる．ただし，システムコールを呼び出す際に指定されたFCBにセットされているドライブ番号が00H以外であった場合には，本システムコールによってセットされているデフォルト・ドライブの設定は無視される．

● ログイン・ベクトルの獲得

ファンクション：18H

設定：なし

戻り値：HLレジスタ←オンライン・ドライブ情報

オンライン・ドライブとは，MSXに正常な状態で接続され，かつその電源が投入されているドライブを指す．このシステムコールを実行すると各ドライブがオンラインであるか否かを調べ，

その結果を**図3.22**のようにHLレジスタに入れて返す．それぞれのビットが“1”ならば対応するドライブはオンラインであり，“0”ならばそうでないことを示す．

レジスタ名	H								L							
ビット番号	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ドライブ名	MSX-DOSでは意味を持たない								H:	G:	F:	E:	D:	C:	B:	A:
オンライン/オフライン	各ビットについて，オンなら1が，オフなら0がセットされる．															

図3.22　ログイン・ベクトル

● デフォルト・ドライブ番号の獲得

ファンクション：19H

設定：なし

戻り値：Aレジスタ←デフォルト・ドライブ番号(A=00H, B=01H, …)

● 転送先アドレスの設定

ファンクション：1AH

設定：DEレジスタ←設定する転送先アドレス(DMAアドレス)

戻り値：なし

DMAアドレスは，システムリセット時に0080Hに初期化されるが，このファンクション1AHのシステムコールを用いることによって，任意のアドレスに設定し直すことができる．

● ディスク情報の獲得

ファンクション：1BH

設定：Eレジスタ←目的のディスクが入っているドライブ番号
(デフォルト・ドライブ=00H, A=01H, B=02H…)

戻り値：Aレジスタ←1クラスタあたりの論理セクタ数
(Eレジスタの設定が不適当だった場合FFH)
BCレジスタ←論理セクタのサイズ
DEレジスタ←クラスタの総数
HLレジスタ←未使用クラスタの総数
IXレジスタ←DPBの先頭アドレス
IYレジスタ←メモリ上のFATの先頭アドレス

指定ドライブのディスクの情報を得るシステムコールである．ドライブ番号に00Hを指定する

とデフォルト・ドライブの指定になる．それ以外は，Aドライブなら01H，Bドライブなら02H，…を指定する．

このシステムコールはMSX-DOS用に変更されたもので，CP／Mとの互換性はない．

● 日付の獲得

ファンクション：2AH

設定：なし

戻り値：HLレジスタ←年

Dレジスタ←月

Eレジスタ←日

Aレジスタ←曜日

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● 日付の設定

ファンクション：2BH

設定：HLレジスタ←年

Dレジスタ←月

Eレジスタ←日

戻り値：設定が成功したかどうかをAレジスタにセットする．成功なら00Hが，失敗なら0FFHがセットされる．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● 時刻の獲得

ファンクション：2CH

設定：なし

戻り値：Hレジスタ←時

Lレジスタ←分

Dレジスタ←秒

Eレジスタ←1／100秒

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● 時刻の設定

ファンクション：2DH

設定：Hレジスタ←時
Lレジスタ←分
Dレジスタ←秒
Eレジスタ←1／100秒

戻り値：成功なら00Hが，失敗なら0FFHがAレジスタにセットされる．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● ベリファイ・フラグの設定

ファンクション：2EH

設定：ベリファイ・フラグをリセットする時には，Eレジスタ←00H
ベリファイ・フラグをセットする時には，Eレジスタ←00H以外の値

戻り値：なし

ベリファイ・フラグを設定すると，以後のディスクに対する書き込みがベリファイ付きで行われるようになる．つまり，書き込んだ後でその内容をディスクから読み出して，書き込むべき内容と比較して等しいかどうかをチェックする．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

4.3　アブソリュートREAD／WRITE(セクタの直接アクセス)

MSXでは，“論理セクタ”と呼ばれるものを最小の単位としてディスクを管理しています．論理セクタとは，ディスクの物理的なセクタとは無関係に定められたセクタであり，第0論理セクタから最大論理セクタ(ディスクの種類によってその数は異なる)までの一連の番号が付けられています．

この論理セクタのおかげで，MSX-DOSやMSX DISK-BASICのユーザーは，ディスクのメディアタイプによって異なる1トラックごとの物理セクタ数の違いを考慮せずにディスクをアクセスすることが可能となっています．実際にはFCB(ファイル・コントロール・ブロック)を介するシステムコールを用いることで，論理セクタさえも意識しない手軽でキメ細かなファイルの取り扱いができますから，この論理セクタを実際に使用する必要はありません．しかし，論理セクタを用いたアクセスも用途によっては必要であり，MSX-DOSおよびMSX DISK-BASICでは，論理セクタのアクセスするシステムコールも用意しています．

本節では，この論理セクタのアクセスを行うシステムコールを説明します．

● 論理セクタを用いた読み出し

ファンクション：2FH

設定：読み出しを開始する論理セクタの番号(複数の場合はその先頭の論理セクタ番号)をDEレジスタにセットする．読み出す論理セクタの個数をHレジスタに，読み出すディスクのドライブ番号(A：00H，B：01H…)をLレジスタにセットする．

戻り値：読み込んだ内容をDMAバッファにセットする．

指定ドライブの指定論理セクタから，指定された個数の(連続した)論理セクタを読み出し，その内容をDMA以降のメモリに格納する．その際，格納されるメモリ領域は，ファンクション1AH(転送先アドレスの設定)によって充分な量(読み出す論理セクタの個数×論理セクタのサイズバイト以上)を確保しておく必要がある．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● 論理セクタを用いた書き込み

ファンクション：30H

設定：書き込む内容をDMAで示されるアドレス以降のメモリ領域にセットする．書き込みを始める論理セクタの番号を，DEレジスタにセットする．書き込む論理セクタの個数をHレジスタにセットする．書き込むドライブ番号(A：00H，B：01H…)をLレジスタにセットする．

戻り値：なし

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

リスト3.4　クラスタ・ダンプ

```
;==========================================================
;
;   List 3.4     cluster dump
;
;                this program must link List 3.3
;
;==========================================================
;
        EXTRN   GETARG
        EXTRN   STOHEX
        EXTRN   PUTHEX
        EXTRN   PUTCHR
        EXTRN   DUMP8B

BDOS    EQU     0005H

;----- program start -----

        LD      A,1
        CALL    GETARG          ;[DE] := 1st argument of command line
        CALL    STOHEX          ;HL := evaluate [DE] as hexadecimal
                                ;       this is target cluster No.
```

コマンドラインで指定した任意クラスタの先頭の128バイトをダンプする．

```
        PUSH    HL
        LD      E,00H           ;request the default drive
        LD      C,1BH           ;get disk information
        CALL    BDOS
        POP     HL
        CP      0FFH            ;fail ?
        JR      NZ,L2           ;if not fail, A := sector/cluster and goto L2

        LD      DE,ERMSG1       ;[DE] := 'Cannot get Disk information.'
        LD      C,09H           ;string output function
        CALL    BDOS
        RET                     ;error return

L2:     LD      E,(IX+12)       ;DE := 1st sector of data area
        LD      D,(IX+13)
        DEC     HL
        DEC     HL              ;HL := cluster No. - 2
        LD      B,H
        LD      C,L             ;BC := cluster No. - 2
LOOP:   DEC     A               ;Count N times
        JR      Z,RESULT
        ADD     HL,BC
        JR      LOOP
RESULT: ADD     HL,DE           ;HL := sector of terget cluster
        PUSH    HL              ;save target sector
        LD      DE,NEWDMA       ;we reserved 1024 bytes area for DMA
        LD      C,1AH           ;Set DMA address function
        CALL    BDOS
        LD      C,19H
        CALL    BDOS            ;default drive ?
        LD      L,A
        POP     DE              ;DE := target sector
        LD      H,1             ;H := 1 (read 1 secter only)
        LD      C,2FH           ;abusolute read function
        CALL    BDOS            ;data will be set into DMA

DUMP:   LD      HL,NEWDMA       ;HL := DMA address
        LD      DE,0000H        ;DE := relative address from cluster top
        LD      B,16            ;dump 16 lines
DLOOP:  PUSH    BC
        LD      A,D
        CALL    PUTHEX
        LD      A,E
        CALL    PUTHEX
        LD      A,' '
        CALL    PUTCHR
        PUSH    HL
        LD      HL,8
        ADD     HL,DE
        EX      DE,HL           ;DE := DE+8
        POP     HL
        CALL    DUMP8B          ;8 bytes dump subroutin (in another file)
        POP     BC
        DJNZ    DLOOP
        RET                     ;all work have done.

;----- work area -----

NEWDMA: DS      1024            ;Private DMA area
ADRS:   DS      2

ERMSG1: DB      'Cannot get Disk information.$'
ERMSG2: DB      'Cannot read that cluster.$'

        END
```

4.4 FCBを用いたファイルアクセス

前節で述べた論理セクタを直接読み書きするタイプのシステムコールでは，ファイル単位のアクセスが非常に困難であり，ファイルを基本としたディスクアクセスを行うためには，FCBを介するシステムコールを使用する必要があります．

FCBを用いるシステムコールには，大きく分けて次の3つの種類が存在します．1つはシーケンシャル・ファイルアクセスであり，もう1つはランダム・ファイルアクセスです，この2つはCP／Mとの互換性を保つために用意されたシステムコールであり，MSX-DOSが持っている力を充分に引き出すものとはいえません．それに対して，3つ目のランダム・ブロック・アクセスはMSXシステムに独自のシステムコールになっています．このランダム・ブロック・アクセスは，次のような特徴を持っています．

・レコードサイズを任意に設定できる
・複数のレコードをランダムにアクセスできる
・ファイルの大きさをバイト単位で管理できる

本節では，このランダム・ブロック・アクセスを含む，FCBを利用したファイルアクセスのためのシステムコールを説明します．

● ファイルのオープン

ファンクション：0FH

設定：DEレジスタ←オープンされていないFCBの先頭アドレス

戻り値：ファイルのオープンが成功すれば00Hを，失敗すれば0FFHをAレジスタにセットする．ファイルのオープンが成功した場合には，指定されたFCBの各フィールドの設定が行われる

ドライブ番号に00Hをセットすると，ファンクション0EH(デフォルト・ドライブの設定)によって設定されているデフォルト・ドライブによる指定が有効になる．デフォルト・ドライブ以外のファイルをオープンする場合には，Aドライブなら01H，Bドライブなら02H，…のように指定する．

このシステムコールを利用してファイルをオープンすると，ディスク上のディレクトリ領域の情報から，レコードサイズ，カレントブロック，カレントレコード，ランダムレコードの各フィールドを除く，すべてのFCBのフィールドが設定される．設定されないフィールドについては，本システムコール実行後にユーザーが必要に応じて設定する必要がある．FCBの各フィールドがセットされたこの状態が，FCBを介するシステムコールを利用するユーザーにとって真に

“ファイルがオープンされた状態”であり，以下に説明する，FCBを用いたファイルアクセスを行うシステムコールが利用できるようになる．

● ファイルのクローズ

ファンクション：10H

設定：DEレジスタ←オープンされたFCBの先頭アドレス

戻り値：ファイルのクローズが成功すれば00Hを，失敗すれば0FFHをAレジスタにセットする．さらにFCBのドライブ番号を設定する．

現在のメモリ上のFCBの内容をディスク上の該当するディレクトリ・エリアに書き込むことによって，ファイルの更新に関する整合性を保つ．ファイルに対して読み込みしか行っていない場合には，このシステムコールを使ってファイルをクローズする必要はない．

● ファイルの検索　その1

ファンクション：11H

設定：DEレジスタ←オープンされていないFCBの先頭アドレス

戻り値：ファイルが見つかった場合には00Hを，見つからなかった場合には0FFHをAレジスタにセットする．見つかった場合にはさらに，DMAで示される領域にそのファイルのディスク上のディレクトリ・エントリ(32バイト)をセットする．また，FCBのドライブ番号もセットする．

ファイル名にはワイルドカード・キャラクタを使用することができる．たとえば，ワイルドカード・キャラクタを使って“????????.c”という指定を行うと，「先頭の文字列は何でもよいが，末尾に“.c”が付くファイル名」を検索し，最初にマッチしたファイルのディレクトリ情報をDMA以降に書き込む．マッチするすべてのファイルを検索したい場合や，マッチするファイルが1個だけかどうかを知りたい場合には，次に紹介するファンクション12Hを利用する必要がある．

● ファイルの検索　その2

ファンクション：12H

設定：なし

戻り値：見つかった場合には00Hを，見つからなかった場合には0FFHをAレジスタにセットする．見つかった場合にはさらに，DMAで示される領域にそのファイルのディレクトリ・エントリ(32バイト)をセットする

このシステムコールは，ファンクション11Hのワイルドカード・キャラクタによるファイル名の指定とマッチした複数のファイルを検索する時に使用するもので単独で使用しても意味がない．

このシステムコールを使用すると，ファンクション11Hによってマッチした複数のファイルのディレクトリ情報を，1つずつ順番に得ていくことができる．

● ファイルの抹消

ファンクション：13H

設定：DEレジスタ←オープンされたFCBの先頭アドレス

戻り値：抹消が成功した場合には00Hを，失敗した場合には0FFHをAレジスタにセットする

ファイル名にワイルドカード・キャラクタを使用して，複数のファイルを一度に抹消することができる．

● シーケンシャルな読み出し

ファンクション：14H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのカレントブロック←読み出しを開始するブロック
FCBのカレントレコード←読み出しを開始するレコード

戻り値：読み込みが成功すれば00Hを，失敗すれば01HをAレジスタにセットする．成功した場合には，読み込んだ1レコードをDMAで示される領域にセットする

FCBのカレントブロックとカレントレコードは，読み込み後に自動的に更新される．つまり，連続して読み込む場合には，カレントブロックとカレントレコードをセットする必要はない．読み込むレコードサイズは128バイト固定．

● シーケンシャルな書き込み

ファンクション：15H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのカレントブロック←書き込みを開始するブロック
FCBのカレントレコード←書き込みを開始するレコード
DMA以降の128バイト←書き込むデータ

戻り値：書き込みが成功すれば00Hを，失敗すれば01HをAレジスタにセットする．

FCBのカレントブロックとカレントレコードは，書き込み後に自動的に更新される．

● ファイルの作成

ファンクション：16H

設定：DEレジスタ←オープンされていないFCBの先頭アドレス

戻り値：ファイルの作成が成功すれば00Hを，失敗すれば0FFHをAレジスタにセットする

FCB中のレコードサイズ，カレントブロック，カレントレコード，ランダムレコードの各フィールドについては，これらを本システムコールの実行後に必要に応じてセットしておく必要がある．

● ファイル名の変更

ファンクション：17H

設定：旧ファイル名に対応するFCBの18バイト目(FCBのファイルサイズ・フィールドの2バイト目＝旧ファイル名の16バイト後方)からの11バイトに(18〜28バイト)新ファイル名をセットし，そのFCBのアドレスをDEレジスタにセットする

戻り値：ファイル名の変更が成功すれば00Hを，失敗すれば0FFHをAレジスタにセットする

新旧ファイル名に，ワイルドカード・キャラクタ"?"を使用できる．たとえば，旧ファイル名に"????????.o"を，新ファイル名に"????????.obj"を指定すれば，".o"という拡張子を持つすべてのファイルについて，拡張子を".obj"に変更することができる．

● ランダムな読み出し

ファンクション：21H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのランダムレコード←読み出すレコード

戻り値：読み込みが成功すれば00Hを，失敗すれば01HをAレジスタにセットする．読み込みが成功した場合には，DMAで示される領域に読み込んだ1レコードの内容をセットする

レコードの大きさは128バイト固定長．

● ランダムな書き込み

ファンクション：22H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのランダムレコード←書き込むレコード
DMA以降の128バイト←書き込むデータ

戻り値：書き込みが成功すれば 00Hを，失敗すれば 01HをAレジスタにセットする

レコードの大きさは128バイト固定長.

● ファイルサイズの獲得

ファンクション：23H

設定：DE レジスタ←オープンされた FCB の先頭アドレス

戻り値：ファイルサイズの獲得が成功すれば00Hを，失敗すれば 0FFH をAレジスタにセットする．獲得が成功すれば，FCB のランダムレコード・フィールドに，指定されたファイルの128バイト単位のサイズをセットする.

ファイルのサイズは128バイト単位で計算されます．つまり，200バイトであれば2が，257バイトであれば3がセットされます.

● ランダムレコード・フィールドの設定

ファンクション：24H

設定：DE レジスタ←オープンされた FCB の先頭アドレス
FCB のカレントブロック←目的のブロック
FCB のカレントレコード←目的のレコード

戻り値：ランダムレコード・フィールドに，指定された FCB のカレントブロック・フィールドとカレントレコード・フィールドから計算したカレントレコード・ポジションをセットする

● ランダムな書き込み　その2(ランダム・ブロック・アクセス)

ファンクション：26H

設定：DE レジスタ←オープンされた FCB の先頭アドレス
FCB のレコードサイズ←書き込むレコードサイズ
FCB のランダムレコード←書き込みを開始するレコード
HL ←書き込むレコード数
DMA 以降のメモリ領域←書き込むデータ

戻り値：データの書き込みに成功すれば 00H，失敗すれば 01H をAレジスタにセットする

書き込みが終わると，ランダムレコード・フィールドの値が自動的に更新されて，最後に書き込んだレコードの次のレコードを指す．ひとつのレコードの大きさはFCBのレコードサイズ・フィールドによって1バイトから65535バイトまで任意に設定できる．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● ランダムな読み出し　その2(ランダム・ブロック・アクセス)

ファンクション：27H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのレコードサイズ←読み出すレコードサイズ
FCBランダムレコード←読み出しを開始するレコード
HL←読み出すレコード数

戻り値：データの読み出しに成功すれば 00H，失敗すれば 01H をAレジスタにセットする．さらにHLレジスタに，実際に読み込んだレコードの個数をセットする

読み込みが終わると，ランダムレコード・フィールドの値が自動的に更新される．このシステムコール実行後，HLレジスタには実際に読み込んだレコードの総数がセットされる．つまり，指定した数のレコードを読み込み終わる前にファイルの終わりに達してしまった場合には，それまでに読み込んだ実際のレコード数がHLレジスタに入ることになる．

このシステムコールはMSX-DOS用に増設されたもので，CP／Mとの互換性はない．

● ランダムな書き込み　その3

ファンクション：28H

設定：DEレジスタ←オープンされたFCBの先頭アドレス
FCBのランダムレコード←書き込むレコード
DMA以降の128バイト←書き込むデータ

戻り値：書き込みが成功すれば 00Hを，失敗すれば 01H をAレジスタにセットします．

レコードの大きさは128バイト固定長．

本システムコールでは，書き込みによってファイルが大きくなる場合，追加されたレコードを00Hで埋めて，指定されたレコードに書き込みを行う．この点を除けばファンクション22H(ランダムな書き込み)と同じである．

リスト3.5 ファイル・ダンプ

```
;============================================================
;
;   List 3.5      file dump
;
;                 this program must link List 3.3
;
;============================================================
;
        EXTRN   GETARG          コマンドラインで指定した任意ファイル をダンプする.
        EXTRN   STOHEX
        EXTRN   PUTCHR
        EXTRN   PUTHEX
        EXTRN   DUMP8B

BDOS    EQU     0005H           コマンドラインの第1パラメータとして指定したファイル
FCB     EQU     005CH           名は, デフォルトFCBエリア(005CH～) に格納される.

;----- program start -----

        LD      DE,FCB          ;DE := default FCB address
        LD      C,0FH           ;open file function
        CALL    BDOS
        OR      A               ;success ?
        JR      Z,READ          ;if so, goto READ

        LD      DE,ERMSG1       ;[DE] := 'Cannot open that file.'
        LD      C,09H           ;string output function
        CALL    BDOS
        RET                     ;error return

READ:   LD      A,2
        CALL    GETARG          ;get 2nd argument of command line
        CALL    STOHEX          ;HL := value of the argument
        LD      (ADRS),HL       ;set address counter

        LD      DE,NEWDMA
        LD      C,1AH           ;set DMA address function
        CALL    BDOS

        LD      HL,8
        LD      (FCB+14),HL     ;record size := 8

        LD      HL,0
        LD      (FCB+33),HL
        LD      (FCB+35),HL     ;random record := 0

RD1:    LD      HL,NEWDMA       ;clear DMA area
        LD      B,8
RD2:    LD      (HL),' '
        INC     HL
        DJNZ    RD2

        LD      HL,1            ;read 1 record
        LD      DE,FCB
        LD      C,27H           ;random block read function
        CALL    BDOS
        OR      A               ;succsess ?
        JR      Z,DUMP          ;if so, goto DUMP

        LD      DE,ERMSG2       ;[DE] := 'Ok.'
        LD      C,09H           ;string output function
        CALL    BDOS
        RET
```

```
DUMP:   LD      HL,(ADRS)
        LD      A,H
        CALL    PUTHEX
        LD      A,L
        CALL    PUTHEX
        LD      A,' '
        CALL    PUTCHR
        LD      DE,8
        ADD     HL,DE
        LD      (ADRS),HL

        LD      HL,NEWDMA
        CALL    DUMP8B          ;dump 8 byte

        JR      RD1

;----- work area -----

ADRS:   DS      2
NEWDMA: DS      8

;----- error message -----

ERMSG1: DB      'Cannot open that file.$'
ERMSG2: DB      'Ok.$'

        END
```

第4部

VDPと画面表示

MSX2の画面表示用LSI（VDP）には，これまでMSXに使用されていたTMS9918Aと上位互換性を持つ，V9938（MSX-VIDEO）が使用されています．MSX2の特徴である優れたグラフィック機能の大部分は，このLSIが担っているといってもよいでしょう．

第4部ではこのMSX-VIDEOの使用法について，BASICではサポートされていない機能まで含め詳しく説明していきます．なお，ハードウェアスペック的な情報など，本書で説明できなかった部分に関して興味をお持ちの方は「V9938 MSX-VIDEOテクニカルデータブック（アスキー出版局発行）」を参照していただくとよいでしょう．

1章　MSX-VIDEOの構成

MSX-VIDEOは次のような特徴を持ち，TMS9918Aと比べて大幅に表示能力が向上しています．

・9ビットのカラーパレットによって，512色の表現が可能
・最大512×424ドットの分解能（インターレース使用時）
・最大256色が同時に表示可能
・グラフィック的に操作しやすいフルビットマップ・モードの採用
・1行80文字のテキスト表示モードの採用
・LINE，SEARCH，AREA-MOVEなどがハードウェア的に実行可能
・スプライトは同一水平ラインに8個まで表示可能
・スプライトは1ラインごとに異なった色の指定が可能
・ビデオ信号のディジタイズ機能内蔵
・スーパーインポーズ機能内蔵

これらの機能を実現するMSX-VIDEOの画面表示システムの構成要素は，数多くのレジスタ群，MSX-VIDEOによって管理されるVRAM，MSX-VIDEOとCPU間のデータ受け渡しを行うI／Oポートの3つの部分に大別されます．

1.1　レジスタ

MSX-VIDEOは，その多くの機能を実現するために，内部に49本のレジスタを備えています．これを本書では“VDPレジスタ”と呼ぶことにします．VDPレジスタは機能によって以下の3種に分類されます．なおコントロール・レジスタ群とステータス・レジスタ群については，BASICからVDP(n)というシステム変数を用いて参照することも可能です．

(1)コントロール・レジスタ群(R#0～R#23，R#32～R#46)

MSX-VIDEOの動作を制御する書き込み専用8ビットレジスタ群です．一般にR#nの記号で表されます．R#0～R#23は主に画面モードの設定に用いられ，R#32～R#46は5章で述べるVDP

コマンドの実行時に用いられます．R#24～R#31に相当する番号のコントロール・レジスタは存在しません．各コントロール・レジスタの役割を**表４.１**に示します．

名称	対応するVDP(n)	機　　能	名称	対応するVDP(n)	機　　能	
R#０	VDP(０)	モードレジスタ#０	R#20	VDP(21)	カラーバースト信号１	
R#１	VDP(１)	モードレジスタ#１	R#21	VDP(22)	カラーバースト信号２	
R#２	VDP(２)	パターンネームテーブル	R#22	VDP(23)	カラーバースト信号３	
R#３	VDP(３)	カラーテーブル(LOW)	R#23	VDP(24)	画面のハードスクロール	
R#４	VDP(４)	パターンジェネレータテーブル				
R#５	VDP(５)	スプライトアトリビュートテーブル(LOW)	R#32	VDP(33)	SX：転送元X座標(LOW)	VDPコマンド用レジスタ
R#６	VDP(６)	スプライトパターンジェネレータテーブル	R#33	VDP(34)	SX：転送元X座標(HIGH)	
R#７	VDP(７)	周辺色／テキストモード時の文字色	R#34	VDP(35)	SY：転送元Y座標(LOW)	
R#８	VDP(９)	モードレジスタ#２	R#35	VDP(36)	SY：転送元Y座標(HIGH)	
R#９	VDP(10)	モードレジスタ#３	R#36	VDP(37)	DX：転送先X座標(LOW)	
R#10	VDP(11)	カラーテーブル(HIGH)	R#37	VDP(38)	DX：転送先X座標(HIGH)	
R#11	VDP(12)	スプライトアトリビュートテーブル(HIGH)	R#38	VDP(39)	DY：転送先Y座標(LOW)	
R#12	VDP(13)	テキストブリンク時の文字色	R#39	VDP(40)	DY：転送先Y座標(HIGH)	
R#13	VDP(14)	ブリンク周期	R#40	VDP(41)	NX：X方向転送ドット数(LOW)	
R#14	VDP(15)	VRAMアクセスアドレス(HIGH)	R#41	VDP(42)	NX：X方向転送ドット数(HIGH)	
R#15	VDP(16)	S#nの間接指定	R#42	VDP(43)	NY：Y方向転送ドット数(LOW)	
R#16	VDP(17)	P#nの間接指定	R#43	VDP(44)	NY：Y方向転送ドット数(HIGH)	
R#17	VDP(18)	R#nの間接指定	R#44	VDP(45)	CLR：対CPUデータ転送用	
R#18	VDP(19)	画面位置の補正(ADJUST)	R#45	VDP(46)	ARG：転送方向／VRAM・拡張RAM指定	
R#19	VDP(20)	割り込み発生の走査線番号	R#46	VDP(47)	CMR：VDPコマンドを発行	

表4.1　コントロール・レジスタ一覧

(2)ステータス・レジスタ群(S#0～S#9)

MSX-VIDEOから得られる各種の情報を読み取るための，読み出し専用８ビットレジスタ群です．一般にS#nという記号で表されます．各レジスタの役割を**表４.２**に示します．

名称	対応するVDP(n)	機　　能	名称	対応するVDP(n)	機　　能
S#０	VDP(８)	割り込み情報	S#５	VDP(−5)	検出したY座標(LOW)
S#１	VDP(−1)	割り込み情報	S#６	VDP(−6)	検出したY座標(HIGH)
S#２	VDP(−2)	VDPコマンド制御情報／その他	S#７	VDP(−7)	VDPコマンドで得られたデータ
S#３	VDP(−3)	検出したX座標(LOW)	S#８	VDP(−8)	サーチコマンドで得られたX座標(LOW)
S#４	VDP(−4)	検出したX座標(HIGH)	S#９	VDP(−9)	サーチコマンドで得られたX座標(HIGH)

表4.2　ステータス・レジスタ一覧

(3)パレット・レジスタ群(P#0〜P#15)

カラーパレットを設定するためのレジスタ群です．一般にP#nという記号で表されます．nはパレット番号を表し，各パレットを512色中の1色に設定する働きをします．1つのパレットレジスタは9ビット長となっており，RGB各色につき，それぞれ3ビットずつ使用されます．

1.2 VRAM

MSX-VIDEOには128Kバイトの VRAM(Video RAM)と64Kバイトの拡張RAMを接続できます．MSX-VIDEOは，この128Kバイトのアドレス空間をアクセスするため，17ビットのアドレス・カウンタを持っています．なお，これらのメモリはMSX-VIDEO自身によって管理されるもので，CPUから直接アクセスすることはできません．

拡張RAMは，VRAMのようにその内容を画面に表示することはできませんが，VDPコマンド実行時にはVRAMと同等に扱うことが可能です．実際，画像データを取り扱う際，これだけの広い領域がワークエリアとして使えると，たいへん便利です．ただし，MSX 2の仕様には拡張RAMを実装しなければならないという項目は含まれていませんから，これを使うと互換性を欠く恐れがあります．

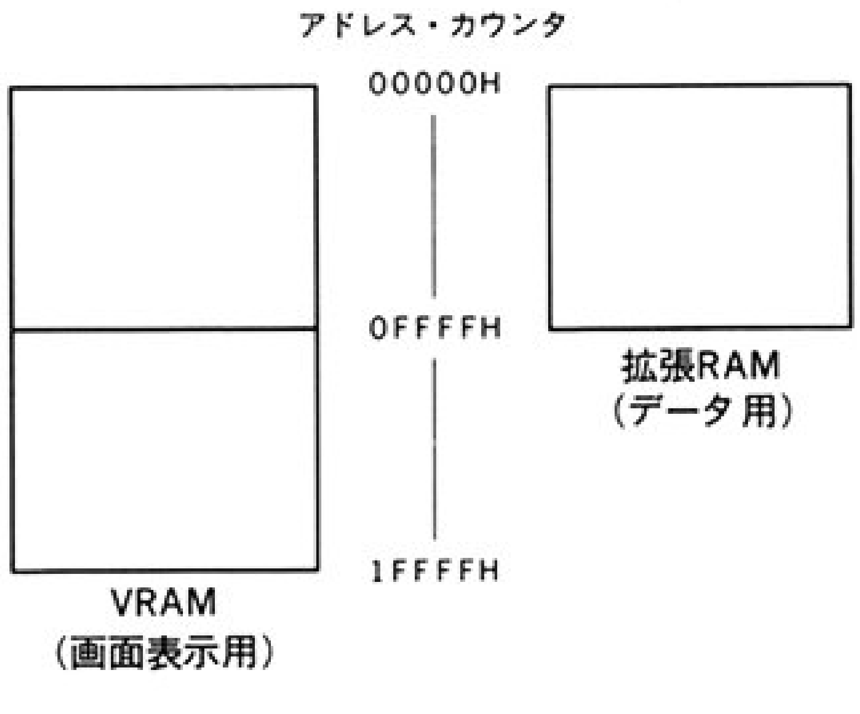


図4.1　VRAMと拡張RAM

1.3 I／Oポート

MSX-VIDEOは外部とデータをやりとりするために，4つのI／Oポートを持っています．これらのポートは**表4.3**に示したような機能を持ち，CPU(Z80)のI／Oアドレスを通して接続されています．表中n，n′で表されたアドレスはMAIN-ROMの6，7番地に記録されています．通常はn＝n′＝98Hですが，機種によっては異なる可能性もありますので，より安全確実なプログラムを目指すためには，この番地の内容を読んでポートアドレスを得なければいけません．

なお，一般に MSX ではプログラムの互換性を確保するために入出力はすべて BIOS を通して実行することが推奨されていますが，画面表示に関しては特にスピードを要求されることが多いため，これらの I／O ポートを介して MSX-VIDEO を直接アクセスすることが許されています．

	アドレス	用　　途
ポート＃0(READ)	n	VRAMからのデータ読み出し
ポート＃0(WRITE)	n′	VRAMへのデータ書き込み
ポート＃1(READ)	n＋1	ステータス・レジスタの読み出し
ポート＃1(WRITE)	n′＋1	コントロール・レジスタへの書き込み
ポート＃2(WRITE)	n′＋2	パレット・レジスタへの書き込み
ポート＃3(WRITE)	n′＋3	間接指定されたレジスタへの書き込み

注） n の値はMAIN-ROMの6番地を参照して得る
n′の値はMAIN-ROMの7番地を参照して得る

表4.3　MSX-VIDEOのポート

2章 MSX-VIDEOのアクセス

前述のように，MSX-VIDEOはI／Oポートを通して直接(BIOSを使わず)アクセスすることが可能です．ここでは，その具体的な方法について説明しましょう．

2.1 レジスタのアクセス

2.1.1 コントロール・レジスタへのデータ書き込み

コントロール・レジスタは書き込み専用のレジスタです．前述のようにBASICからはVDP(n)を参照することによって，一部(R#0～R#23)のコントロール・レジスタの内容を知ることができますが，これはレジスタへの書き込み時にRAM上のワークエリア（F3DFH～F3E6H，FFE7H～FFF6H）に記録しておいた値を読んでいるだけです．

コントロール・レジスタにデータを書き込むには，以下に示す3種類の方法があります．ただし，MSXではスプライトの衝突発生などを調べるために，タイマ割り込みルーチン内でMSX-VIDEOをアクセスしているので，ここに述べた方法を用いてレジスタをアクセスする時には，割り込みを禁止してアクセスの手順が狂わないように注意してください．

(1) 直接指定によるデータ書き込み

第1の方法は，レジスタの直接指定によるアクセスです．**図4.2**のように，ポート#1にまず書き込みたいデータを出力，続けて同じくポート#1に書き込みを行うレジスタ番号（上位2ビットは“10”としておく）を出力することによって行います．

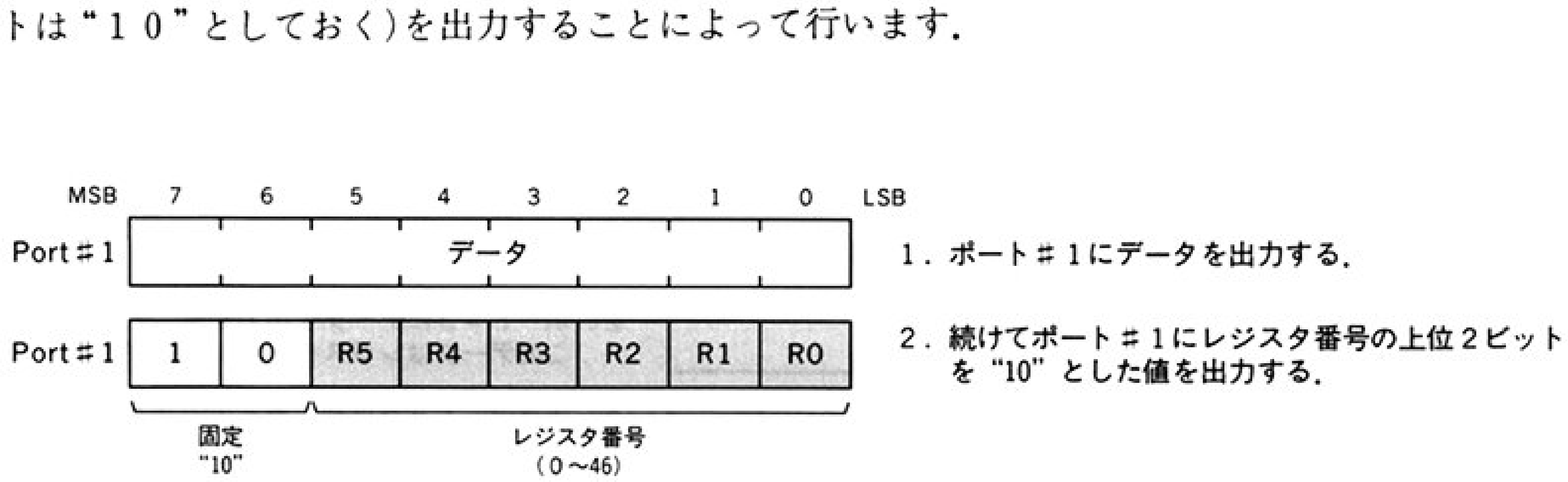


図4.2 直接指定によるR#nのアクセス

ポート#1は，2.2で述べるようにVRAMのアドレスを設定する場合にも用いられますが，どちらの動作を行うかは，このポートに2回目に送られるデータの最上位ビットを参照して選択されます．ここで述べたコントロール・レジスタへのデータ書き込みが実行されるのは，同ビットが“1”の時です．

(2)　間接指定によるデータ書き込み(非オートインクリメント・モード)

第2の方法は，いったん目的とするレジスタの番号をR#17に書き込み，その値によって間接的に指定を行う方法です．まず，直接指定法を用いてR#17(コントロール・レジスタ間接指定レジスタ)にアクセスしたいレジスタの番号を設定します．以後は，ポート#3にデータを出力すれば目的のレジスタにデータを書き込むことができます．この方法はVDPコマンドの実行時など，同じレジスタに連続してデータを送る時に使用します．

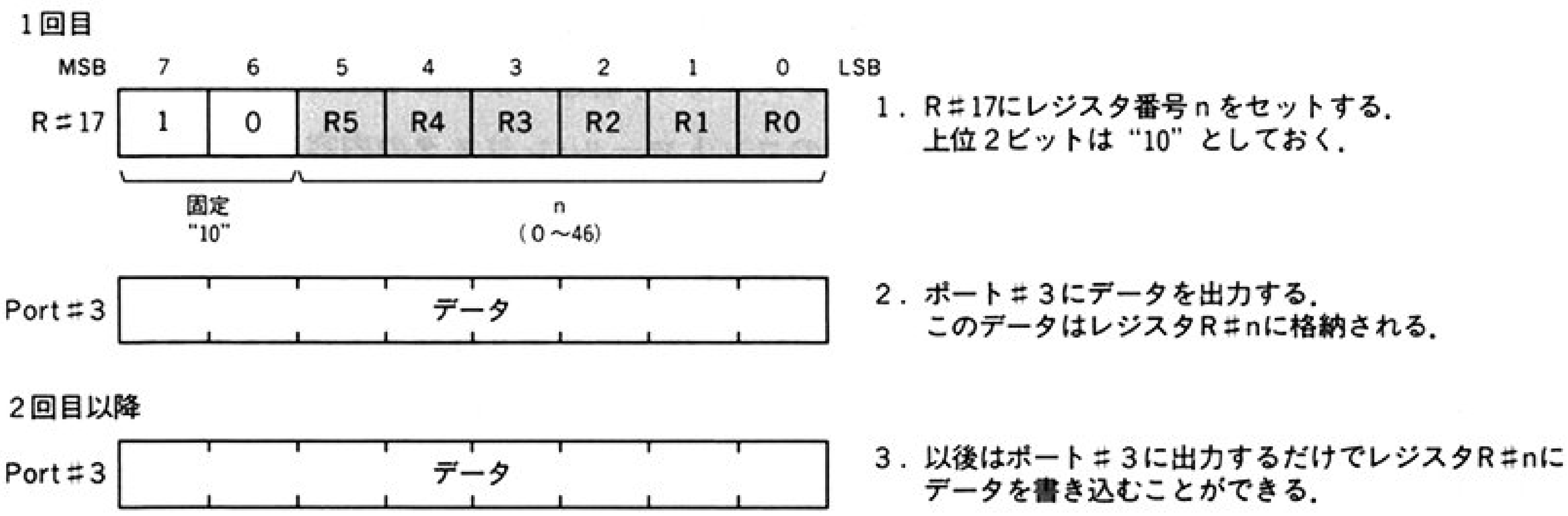


図4.3　間接指定(非オートインクリメント・モード)によるR#nのアクセス

(3)　間接指定によるデータ書き込み(オートインクリメント・モード)

間接指定法において，R#17に設定するレジスタ番号の上位2ビットを“00”としておくと，ポート#3にデータを出力するたびにR#17の内容がインクリメントされます．この方法を用いる

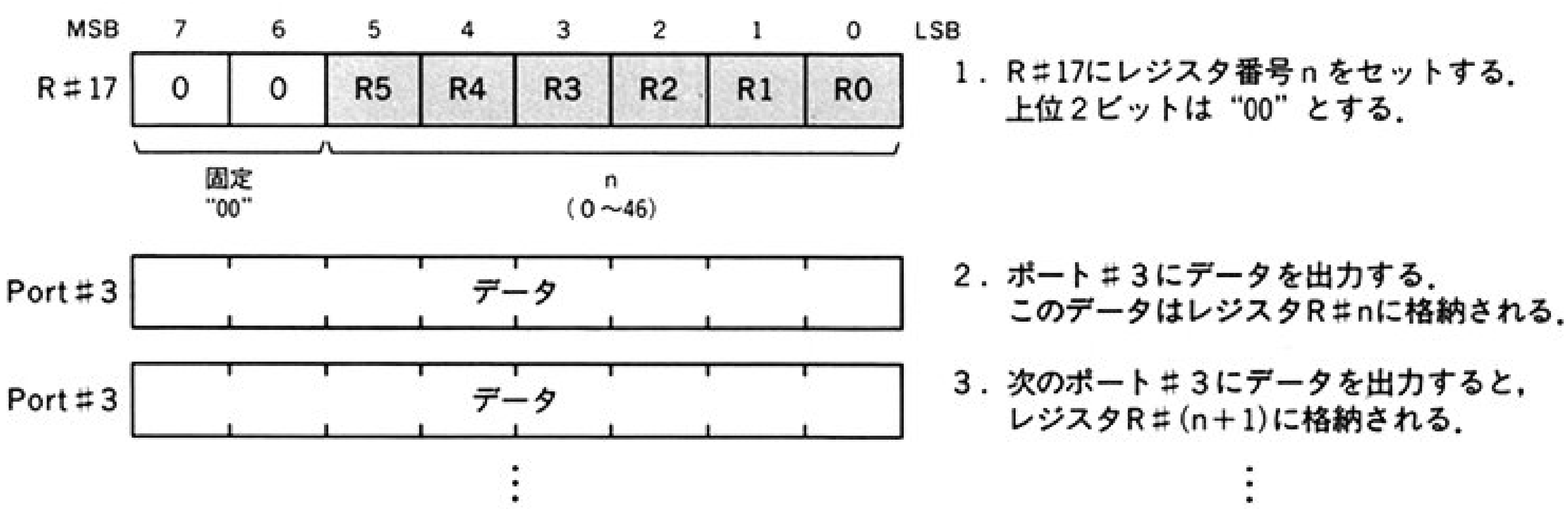


図4.4　間接指定(オートインクリメント・モード)によるR#nのアクセス

と連続するコントロール・レジスタへのデータ書き込みが効率的に行えますから，スクリーンモードの変更時など，連続した多数のレジスタを一度に書き換える場合に役立ちます．

2.1.2 パレットの設定

パレットを設定するには，次のようにR#16(パレット・レジスタ間接指定レジスタ)でパレット番号を指定し，ポート#2へデータを書き込みます．パレット・レジスタは9ビットの長さを持つため，まず赤の輝度と青の輝度，次に緑の輝度と，データは2回に分けて出力します．

ポート#2へ2回データが送られると，R#16は自動的にインクリメントされます．この機能によって，全パレットの初期設定などを簡単に行うことができます．

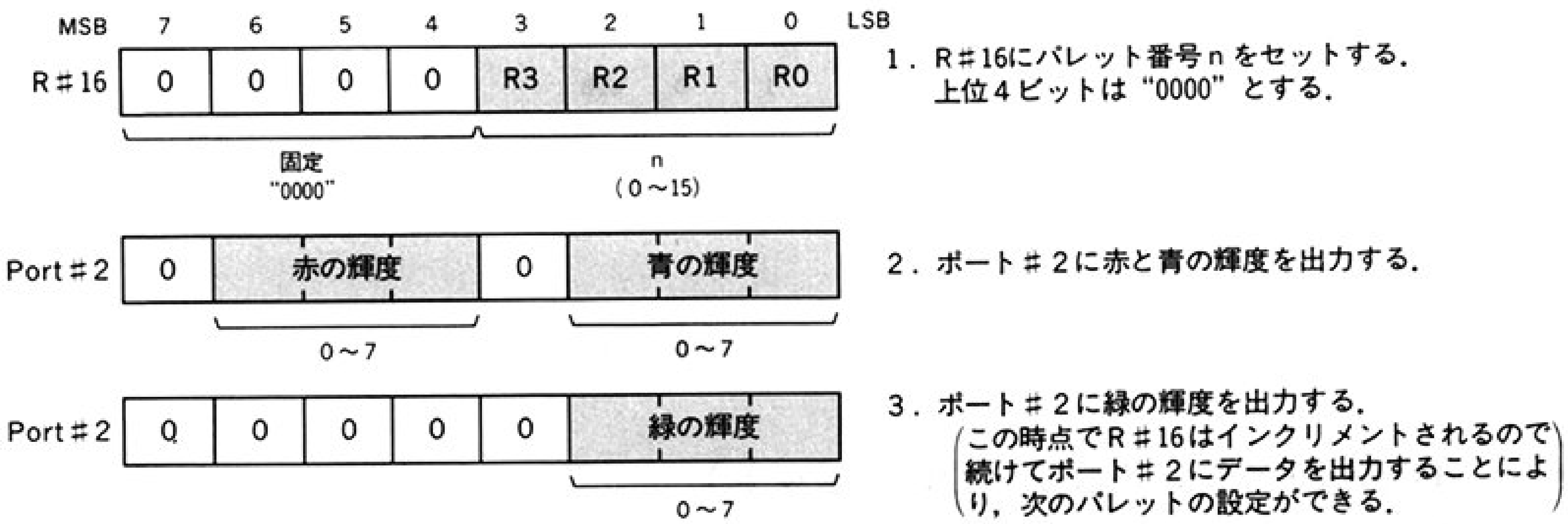


図4.5　パレットの設定

2.1.3 ステータス・レジスタの読み出し

ステータス・レジスタは読み出し専用のレジスタです．図4.6のようにR#15(ステータス・レジスタ間接指定レジスタ)にステータス・レジスタ番号を設定すると，ポート#1からその内容を読み出すことができます．ただしステータスレジスタをアクセスする前にはかならず割り込みを禁止し，そして目的の処理が終了した後でR#15の値を0にしてから割り込みを解除してください．

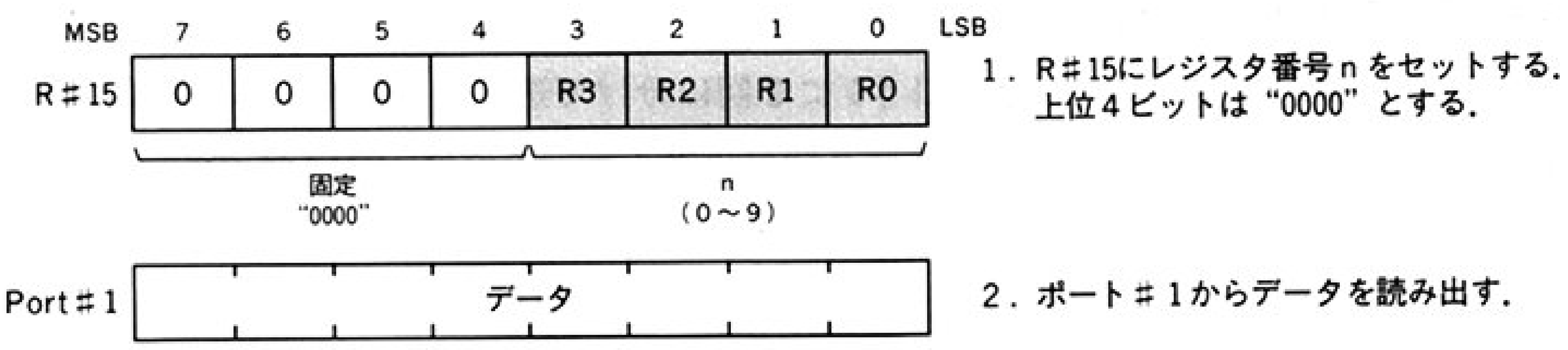


図4.6　ステータス・レジスタのアクセス

2.2　CPU からの VRAM アクセス

VRAM の任意のアドレスを CPU 側からアクセスする場合は，以下のような手順により行います．

①　バンク切り換え

VRAM の前半(00000H～0FFFFH)と拡張 RAM は，MSX-VIDEO から見て同一のアドレスに存在しており，バンク切り換えによって両者を使い分けるようになっています．MSX 2 では拡張 RAM は使用しませんから，常に VRAM のバンクを選択してください．バンクの切り換えはR#45のビット 6 によって行います．

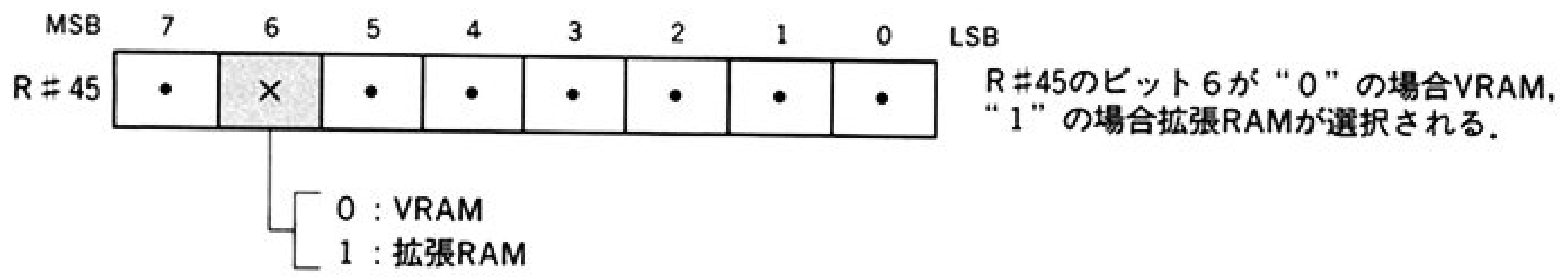


図4.7　VRAM／拡張RAMの選択

②　VRAM のアドレス設定(上位 3 ビット)

128Kバイトの VRAM をアクセスするための17ビットのアドレス情報は，アドレス・カウンタ(A16～A0)に設定されます．アドレス・カウンタの設定は，まず上位 3 ビットから行います．これにはR#14を用います．

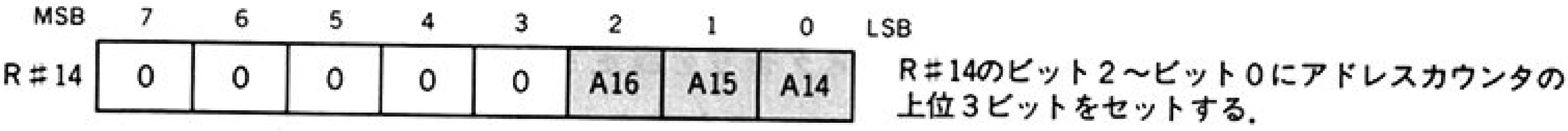


図4.8　上位 3 ビットの設定

③　VRAM のアドレス設定(下位 14 ビット)

次に，アドレスの下位14ビットをポート#1 に 2 回に分けて送り出します．②と③は，かならずこの順序で行わなくてはなりません．この時 2 回目に送るデータには，VRAM のデータを読み出したいのか／ VRAM にデータを書き込みたいのか，という情報も付け加えておきます．

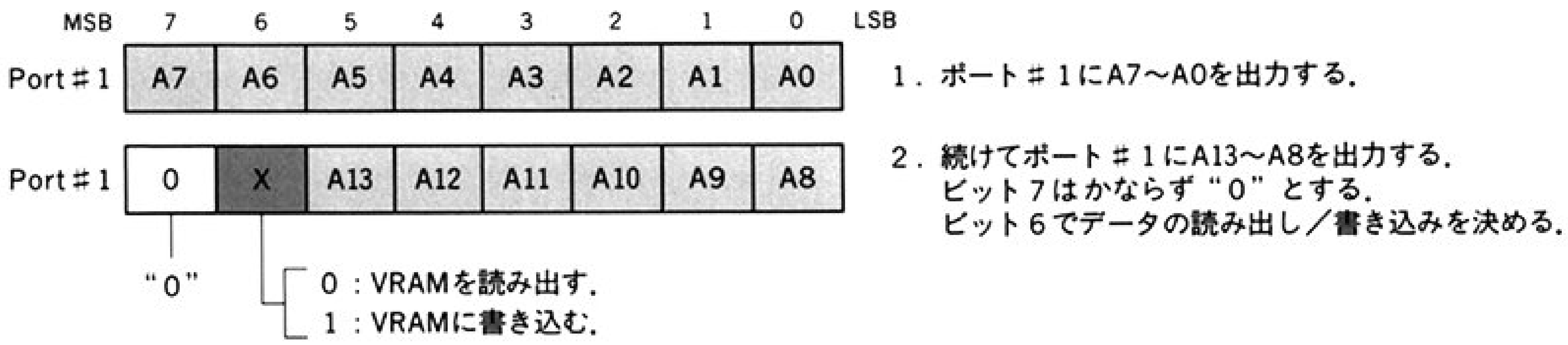


図4.9　下位14ビットとリード／ライト動作の設定

④　VRAMの読み出し／書き込み

アドレス・カウンタに値をセットし終えたら，ポート#0を通してデータの読み書きを行います．なおデータの読み出し／書き込みの指定は，前述のようにアドレス・カウンタのA13～A8を設定する時に同時に行います．

この時，データを1バイト読み書きするごとに，アドレス・カウンタは自動的にインクリメントします．この機能を用いるとVRAMの連続したアドレスを簡単にアクセスできます．

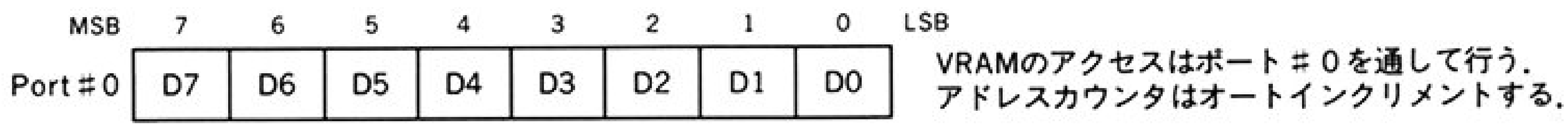


図4.10　ポート＃0を通したVRAMのアクセス

3章 MSX 2の画面モード

MSX 2には，**表4.4**に示す10種の画面モードが存在します．表中で，“*”の印を付けた6つの画面モード(TEXT 2およびGRAPHIC 3～GRAPHIC 7)がMSX 2で新設されたものです．従来からあった他のモードも，TMS9918AからMSX-VIDEOへの変更に伴い，多少機能が増えた部分があります．以下，これら10種類の画面モードの特徴と，その具体的な使用法について説明を行います．

モード名	相当するSCREENモード	特　　徴
TEXT 1	SCREEN 0(WIDTH 40)	1行40文字のテキスト，文字の色は1色
*TEXT 2	SCREEN 0(WIDTH 80)	1行80文字のテキスト，文字はブリンク可能
MULTI COLOR	SCREEN 3	1文字を4ブロック分割した擬似グラフィック
GRAPHIC 1	SCREEN 1	1行32文字のテキスト，色付き文字が可能
GRAPHIC 2	SCREEN 2	256×192，8ドット単位で色を指定する
*GRAPHIC 3	SCREEN 4	スプライトモード2が使えるGRAPHIC 2
*GRAPHIC 4	SCREEN 5	256×212，ドットごとに16色可能
*GRAPHIC 5	SCREEN 6	512×212，ドットごとに4色可能
*GRAPHIC 6	SCREEN 7	512×212，ドットごとに16色可能
*GRAPHIC 7	SCREEN 8	256×212，ドットごとに256色可能

表4.4　MSX2の画面モード一覧

3.1　TEXT1モード

TEXT 1とは以下に示す仕様の画面モードをいいます.

画　面：横40文字×縦24行
背景色／文字色ともに512色中から任意に選択可能
文　字：256種の文字が使用可能
文字の大きさは横6ドット×縦8ドット
1文字ごとにフォントの定義が可能
必要メモリ：文字フォント用…2048バイト(8バイト×256文字)
画面表示用………960バイト(40文字×24行)
BASIC：SCREEN 0(WIDTH 40)に相当

3.1.1　TEXT1モードの設定

TEXT 1に限らずMSX-VIDEOの画面モードの設定は, R#0とR#1の中の5ビットを用いて行います. TEXT 1モードの場合は, **図4.11**に示すとおりです.

	MSB 7	6	5	4	3	2	1	0 LSB
R#0	•	•	•	•	0	0	0	•
R#1	•	•	•	1	0	•	•	•

図4.11　TEXT1モードの設定

3.1.2　TEXT1モードの画面構造

●パターンジェネレータ・テーブル

文字フォントを記憶している領域を, パターンジェネレータ・テーブルと呼びます. パターンジェネレータ・テーブルはVRAM上に置かれ, その先頭から1文字につき8バイトずつ使用して各フォントが定義されていますがフォントの横1ライン中, 下位2ビットは画面に表示されません. したがって, 1文字は6×8のサイズとなります. それぞれの文字フォントには0～255の番号が付けられ, これが文字のキャラクタコードに相当します. 文字を画面に表示する時には, このキャラクタコードで指定します.

パターンジェネレータ・テーブルの位置はR#4で指定します. ただし指定できるのはパターン

ジェネレータ・テーブルの先頭アドレスの上位6ビット(A16～A11)のみであり，アドレスの下位11ビット(A10～A0)は常に“00000000000”であるとみなされます．したがって，パターンジェネレータ・テーブルが設定可能なアドレスは，00000Hから2Kバイト単位の位置になります．BASICからは，このアドレスをBASE(2)というシステム変数を用いて参照できます．**図4.12**はパターンジェネレータ・テーブルの構造です．

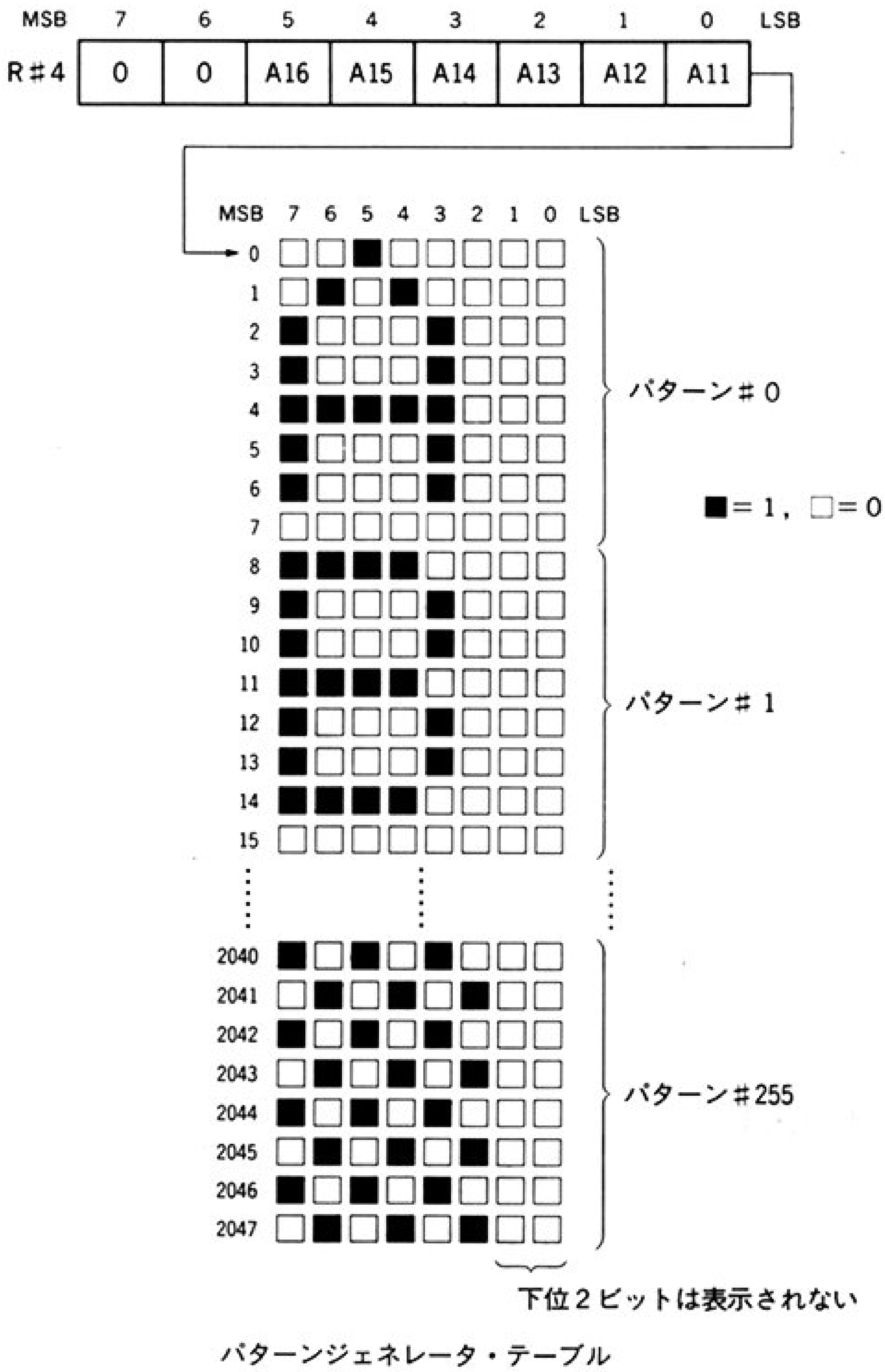


図4.12　パターンジェネレータ・テーブルの構造

●パターンネーム・テーブル

パターンネーム・テーブルとは，1バイトが画面上の1文字に対応するメモリ配列です．このテーブルに0～255のキャラクタコードを書き込むと，対応する画面上の位置にコードで指定した文字が表示されます．

パターンネーム・テーブルの先頭アドレスは，R#2を用いて指定します．指定できるのはアドレスの上位 7 ビット(A16～A10)のみであり，下位 10 ビット(A9～A0)は常に“0000000000”とみなされます．したがって，パターンネーム・テーブルが設定可能なアドレスは，00000Hから 1K バイト単位の位置になります．BASIC からは，このアドレスを BASE(0)というシステム変数を用いて参照できます．**図 4.13** はパターンネーム・テーブルの構造です．

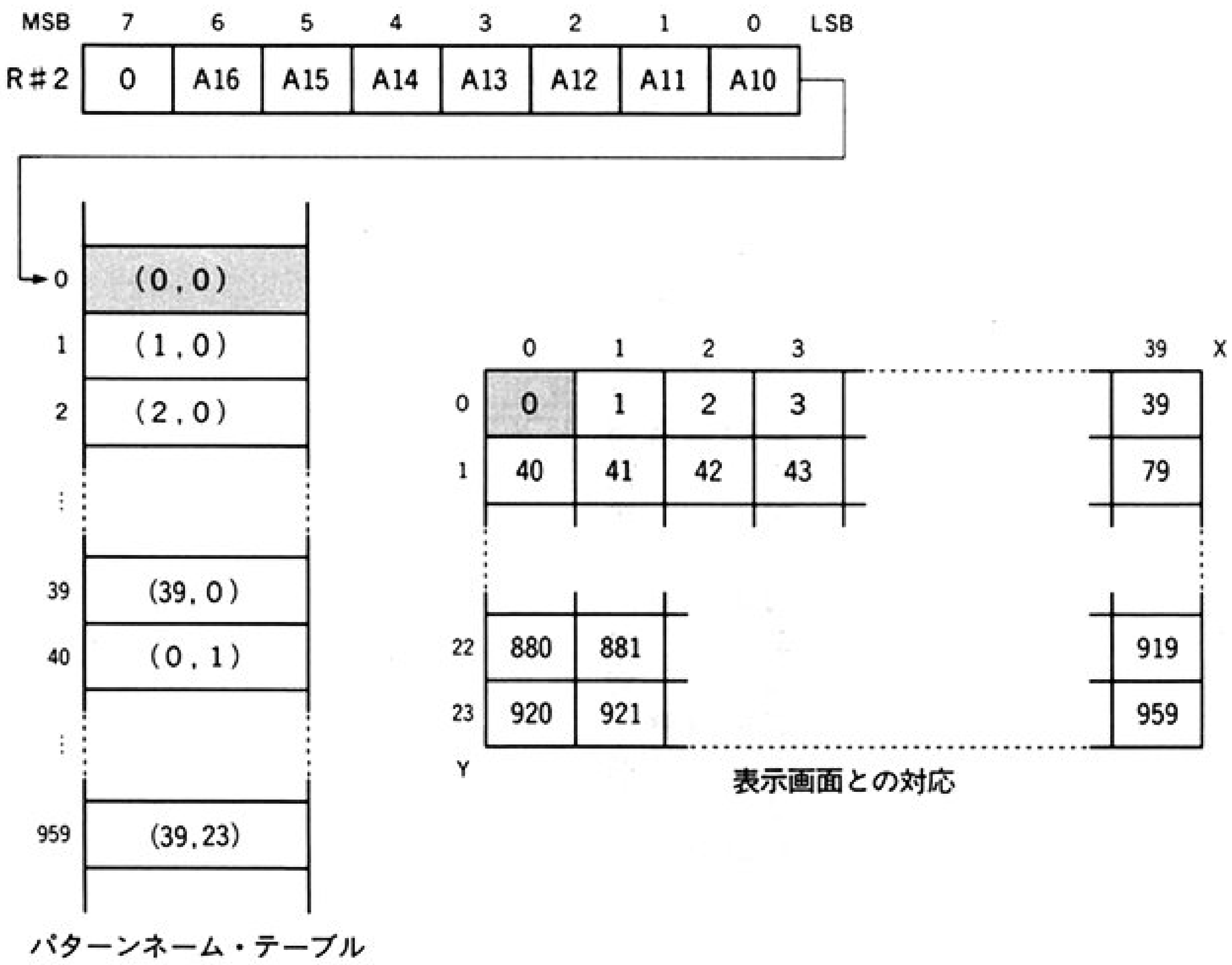


図4.13　TEXT1のパターンネームテーブルの構造

3.1.3　画面の色の指定

画面の色はR#7によって指定します．背景色はR#7の下位 4 ビット，前景色はR#7の上位 4 ビットで指定されたパレットによって色が決まります(**図 4.14**)．フォントパターン中の，“0”の部分は背景色，“1”の部分は前景色で表示されます．なお，TEXT 1 では画面の周辺色を設定することはできず，背景色と同じものになります．



図4.14　TEXT1の色指定

3.2　TEXT 2 モード

TEXT 2 とは以下に示す仕様の画面モードをいいます.

画　面：横 80 文字×縦 24 行(または縦 26.5 行)
　背景色／文字色ともに 512 色中から任意に選択可能
　ブリンク機能の指定により，他に 2 色表示可能

文　字：256 種の文字が使用可能
　文字の大きさは横 6 ドット×縦 8 ドット
　1 文字ごとにフォントの定義が可能
　1 文字ごとにブリンク(点滅)可能

必要メモリ：24行設定時
　　文字フォント用…2048 バイト(8 バイト×256 文字)
　　画面表示用………1920 バイト(80 文字×24 行)
　　ブリンク属性用…240 バイト(＝1920 ビット)
　26.5 行設定時
　　文字フォント用…2048 バイト(同上)
　　画面表示用………2160 バイト(80 文字×27 行)
　　ブリンク属性用…270 バイト(＝2160 ビット)

BASIC：SCREEN 0(WIDTH 80)に相当

3.2.1　TEXT 2 モードの設定

TEXT 2 モードの設定は**図 4.15** のとおりです.

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	0	1	0	•	
R#1	•	•	•	1	0	•	•	•	

図4.15　TEXT 2モードの設定

●行数の設定(24 行／ 26.5 行)

TEXT 2 モードでは，R#9のビット 7 の値によって，1 画面24行または26.5行の切り換えが可能です．ただし，1 画面の行数を26.5行とした場合，最下行に表示した文字はパターンの上半分しか表示されません．この機能は BASIC ではサポートされていません．

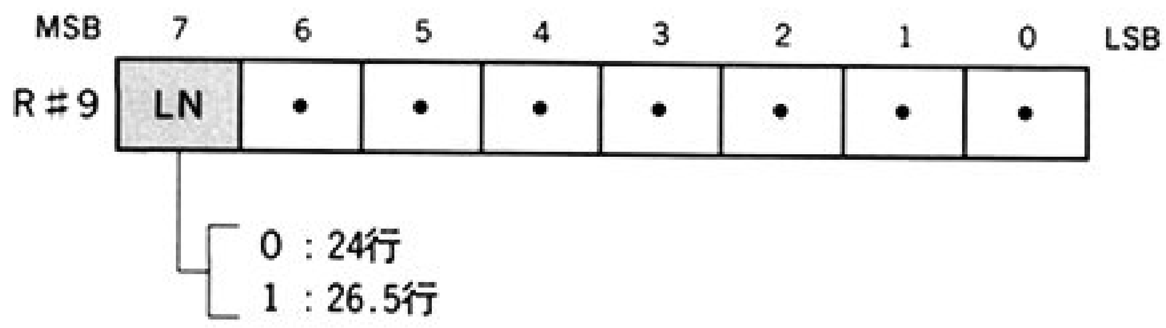


図4.16　行数の切り換え

3.2.2　TEXT 2 の画面構造

● パターンジェネレータ・テーブル

TEXT 1 のパターンジェネレータ・テーブルとまったく同じ構造と機能を持ちます．TEXT 1 の説明を参照してください．

●パターンネーム・テーブル

TEXT 2 では，1 画面に表示される文字が最大2160(＝80×27)文字と増えたため，パターンネーム・テーブルの占める大きさも最大2160バイトになりました．

パターンネーム・テーブルは，先頭アドレスの上位 5 ビット(A16～A12)を R#2に書き込んで位置を設定します．アドレスの下位12ビット(A11～A0)は常に“000000000000”とみなされ，パターンネーム・テーブルが設定可能なアドレスは，00000Hから 4K バイト単位の位置になります．

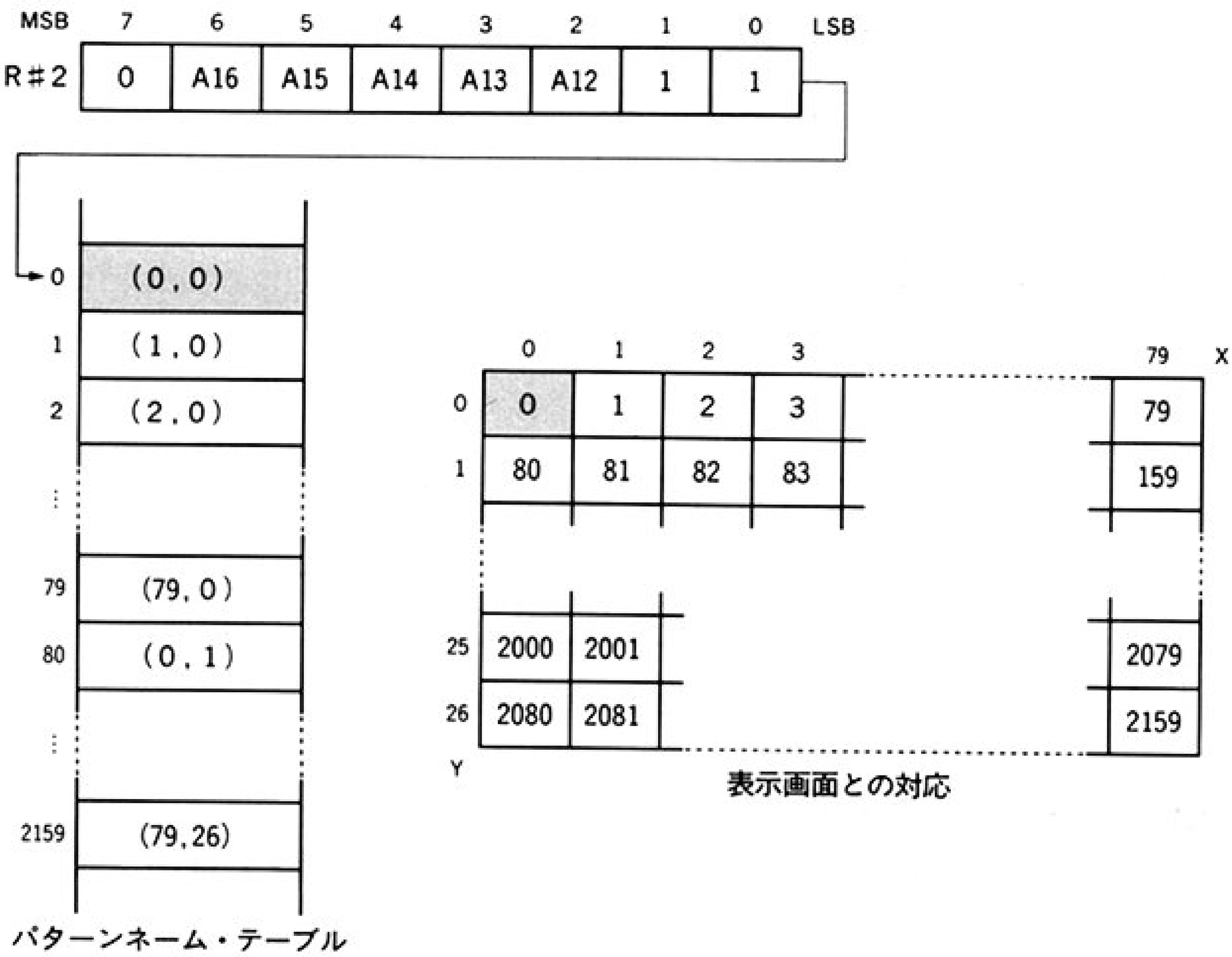


図4.17　TEXT2のパターンネームテーブルの構造

● ブリンクテーブル

TEXT 2 モードでは，文字ごとにブリンク(点滅)させることができます．画面上のどの位置の文字をブリンクさせるかという情報を記憶しておく領域がブリンクテーブルです．ブリンクテーブルは，その1ビットが画面上(すなわちパターンネーム・テーブル上)の1文字に対応しており，対応するビットの内容が“1”である文字にブリンク属性が与えられます．



図4.18　TEXT2のブリンクテーブルの構造

ブリンクテーブルの先頭アドレス指定は，その上位8ビット(A16～A9)をR#3とR#10に設定することによって行います．ブリンクテーブルを設定可能なアドレスは，00000Hから512バイト単位の位置になります．

3.2.3　画面の色と文字のブリンク指定

前景色はR#7の上位4ビット，背景色はR#7の下位4ビットで指定されます．さらに，ブリンクテーブルによってブリンク属性を与えられた文字は，R#7で指定される本来の色と，R#12で指定されるブリンク色とを交互に繰り返します．

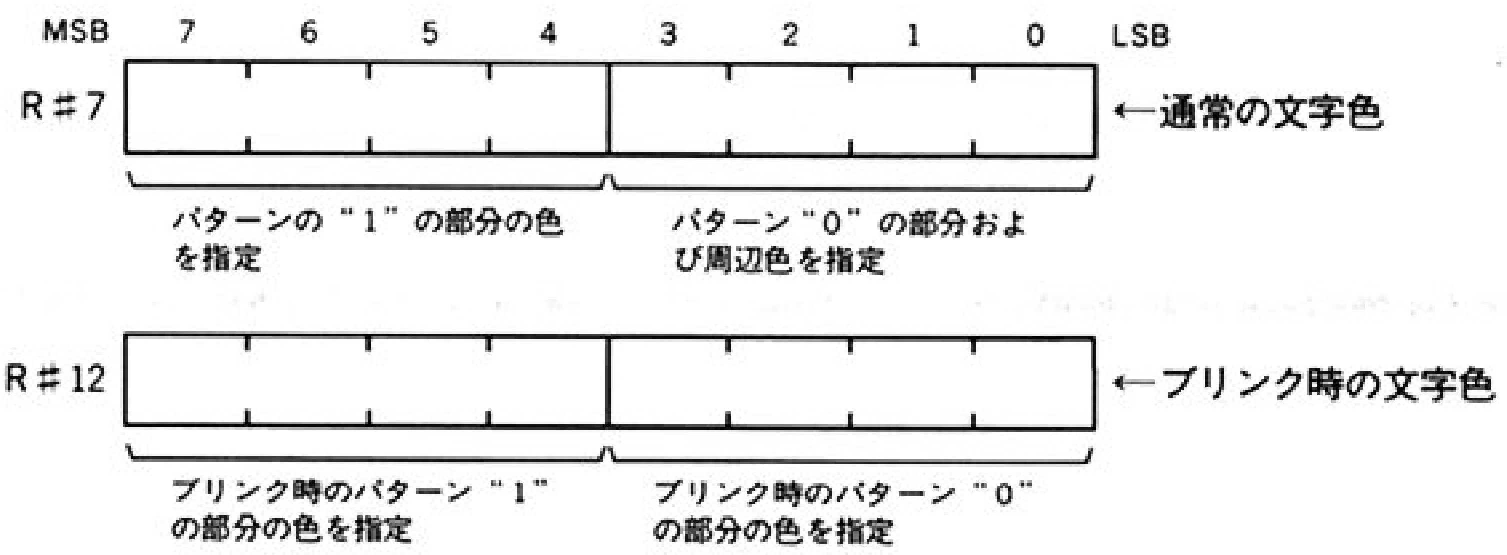


図4.19　画面の色とブリンク色の設定

繰り返しの周期はR#13で設定されます．R#13の上位 4 ビットは本来の文字色が表示されている時間，下位 4 ビットはブリンク色が表示されている時間を 1 / 6 秒単位で表しています．

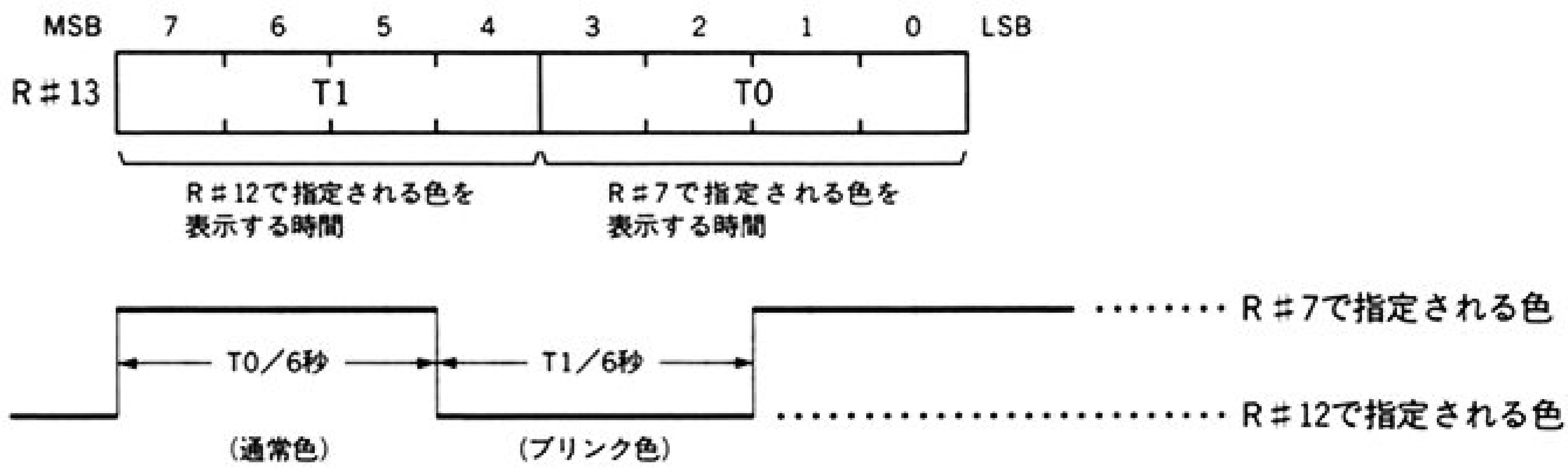


図4.20　ブリンク周期の設定

リスト4.1　ブリンクサンプル

```
1000 '================================================
1010 '  LIST 4.1  BRINK SAMPLE
1020 '================================================
1030 '
1040 SCREEN 0 : WIDTH 80              'TEXT 2 mode
1050 ADR=BASE(1)                     'TAKE COLOR TABLE ADDRESS
1060 '
1070 FOR I=0 TO 2048/8
1080   VPOKE ADR+I,0                 'reset blink mode
1090 NEXT
1100 '
1110 VDP(7) =&HF1                    'text color=15,back color=1
1120 VDP(13)=&H1F                    'text color=1, back color=15
1130 VDP(14)=&H22                    'set interval and start blink
1140 '
1150 PRINT "Input any character : ";
1160 '
1170 K$=INPUT$(1)
1180 IF K$<CHR$(28) THEN 1230
1190 IF K$>" " THEN GOSUB 1280
1200 PRINT K$;
1210 GOTO 1170
1220 '
1230 VDP(14)=0                       'stop blink
1240 END
1250 '
1260 '----- set blink mode -----
1270 '
1280 X=POS(0) : Y=CSRLIN
1290 A=(Y*80+X)¥8
1300 B=X MOD 8
1310 M=VAL("&B"+MID$("0000000010000000",8-B,8))
1320 VPOKE ADR+A,VPEEK(ADR+A) XOR M
1330 RETURN
```

3.3　MULTI COLOR モード

MULTI COLOR モードとは以下に示す仕様の画面モードをいいます．

画　面：横64ブロック×縦48ブロック
512色中の16色が同時に表示可能
ブロック：ブロックの大きさは横 4 ドット×縦 4 ドット
ブロックごとの色指定が可能
必要メモリ：カラー設定用…2048バイト
位置指定用……768バイト
スプライト：スプライトモード 1
BASIC：SCREEN 3 に相当

3.3.1　MULTI COLOR モードの設定

MULTI COLOR モードの設定は，図 4.21 に示すとおりです．

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	0	0	0	•	
R#1	•	•	•	0	1	•	•	•	

図4.21　MULTI COLORモードの設定

3.3.2　MULTI COLOR モードの画面構造

● パターンジェネレータ・テーブル

このモードでは，2 ブロック×2 ブロックで 1 パターンが構成されており，1 個のパターン名には 4 個のパターンが対応しています．パターン中の色は，パターンジェネレータ・テーブルの 2 バイトで表されます．パターンジェネレータ・テーブルの先頭アドレスはR#4で指定します．ただし指定できるのはアドレスの上位 6 ビット(A16～A11)のみであるため，パターンジェネレータ・テーブルに設定可能なアドレスは 00000H から 2K バイト単位の位置となります（図 4.22）．

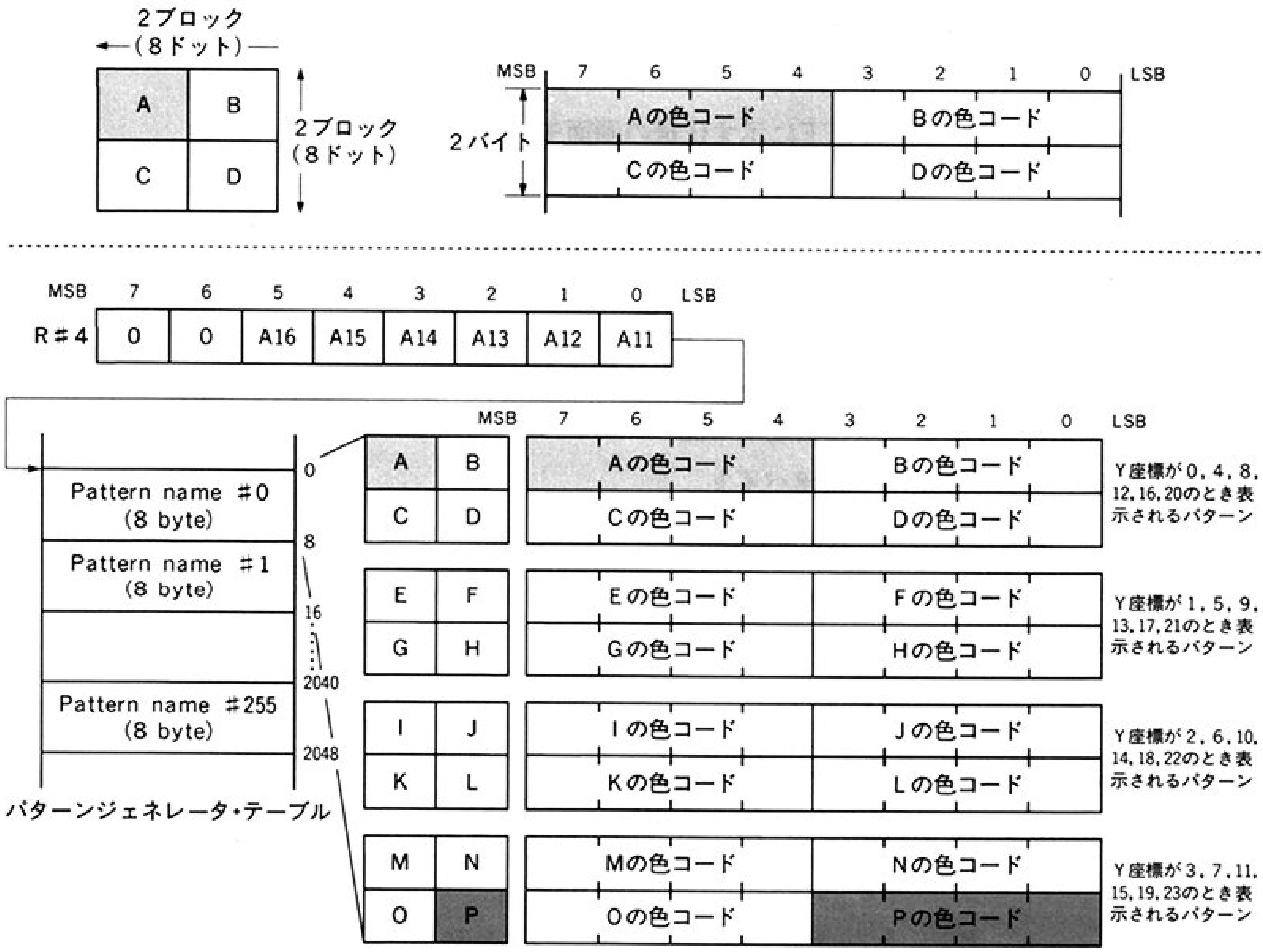


図4.22　MULTI COLORのパターンジェネレータ・テーブルの構造

● パターンネーム・テーブル

画面上の任意の位置に指定のパターンを表示するためのテーブルです．1個のパターン名中の4つのパターンはY座標の値により，そのうちの1つが表示されるようになっています．BASICでは図 4.23 のようにこのテーブルの内容を設定しています．パターンネーム・テーブルの先頭アドレスはR#2で指定します．ただし指定できるのはアドレスの上位7ビット(A16～A10)のみであるため，パターンネーム・テーブルに設定可能なアドレスは00000Hから1Kバイト単位の位置となります(図 4.24)．

3.3.3　MULTI COLORモードの画面の色の指定

R#7で画面の周辺色が指定できます(図 4.25)．

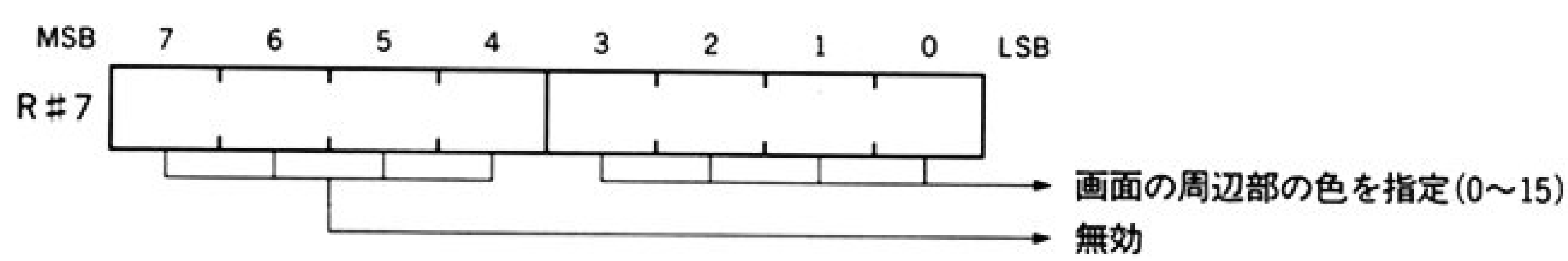


図4.25　周辺色の指定

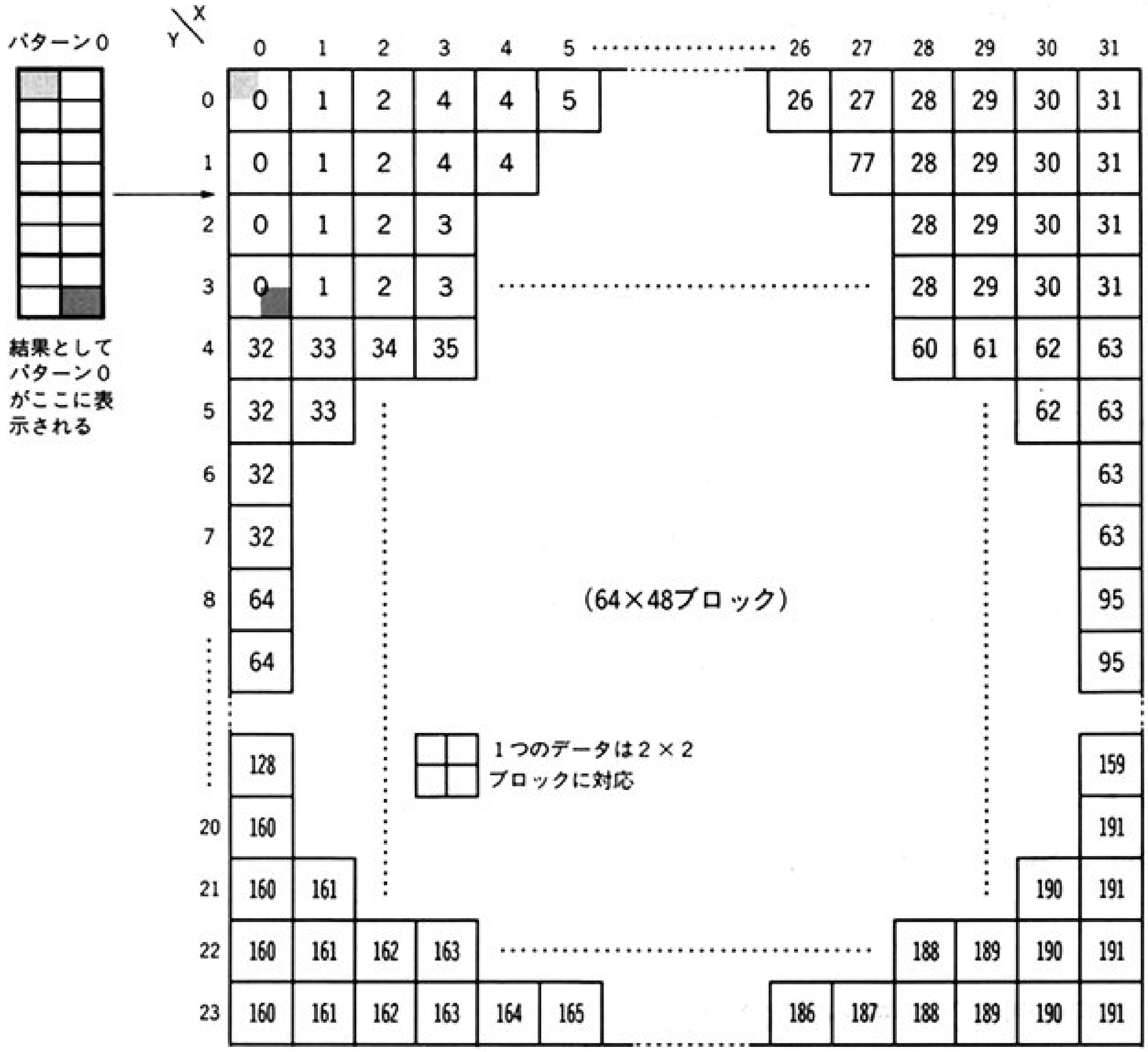


図4.23　BASICのパターンネーム・テーブルの設定

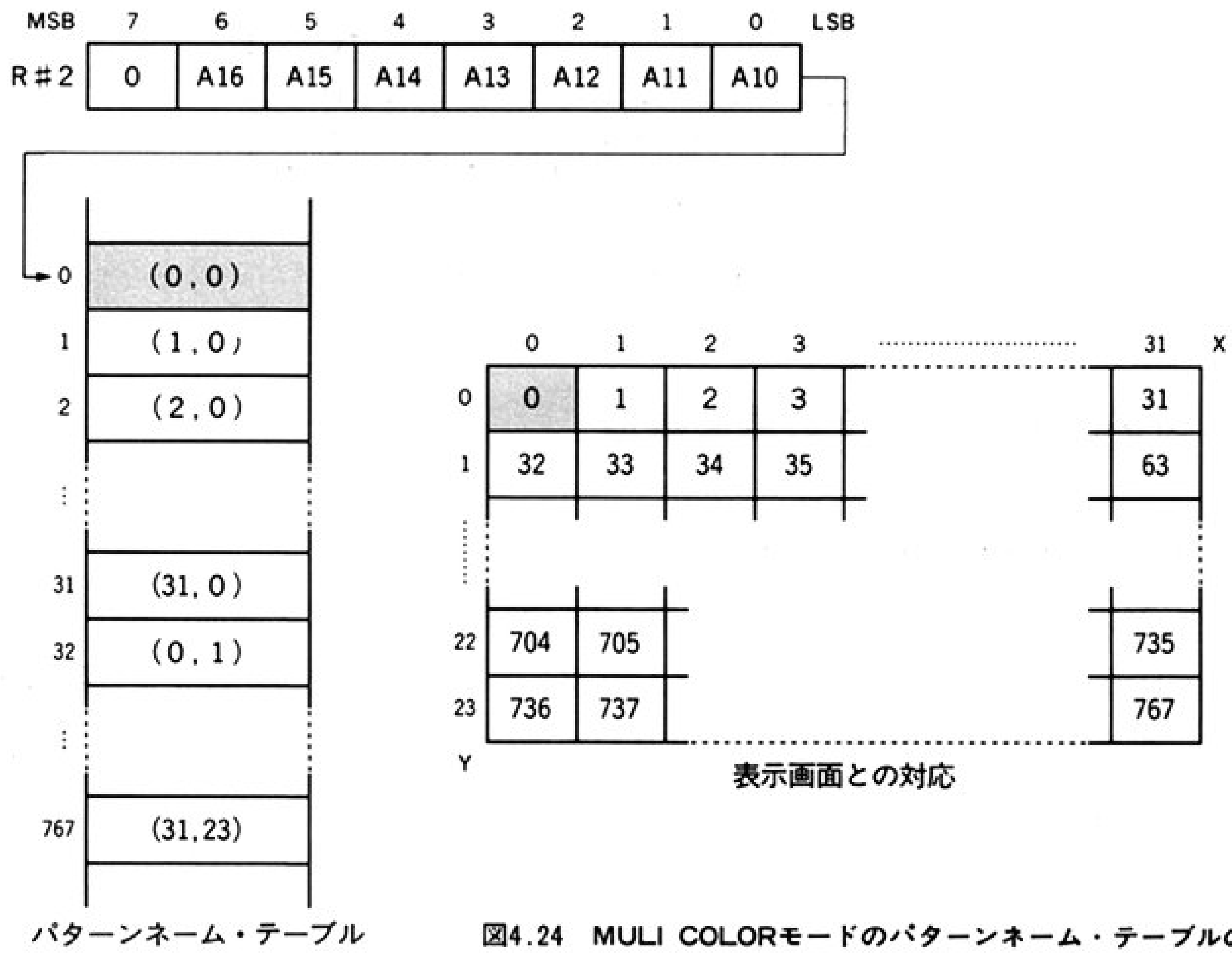


図4.24　MULI COLORモードのパターンネーム・テーブルの構造

3.4　GRAPHIC 1 モード

GRAPHIC1 モードとは以下に示す仕様の画面モードをいいます.

画　面：横32パターン×縦24パターン
512色中の16色が同時に表示可能
パターン：256種類のパターンが使用可能
パターンの大きさは横8ドット×縦8ドット
パターンごとに任意の図形定義が可能
8パターンごとに異なる色が設定可能
必要メモリ：パターンフォント用…2048バイト
カラーテーブル用……32バイト
スプライト：スプライトモード1
BASIC：SCREEN1 に相当

3.4.1　GRAPHIC 1 モードの設定

GRAPHIC 1 モードの設定は，**図4.26** に示すとおりです.

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	0	0	0	•	
R#1	•	•	•	0	0	•	•	•	

図4.26　GRAPHIC 1モードの設定

3.4.2　GRAPHIC 1 モードの画面構造

● パターンジェネレータ・テーブル

このモードでは，0～255のコードに相当する256種類のパターンを画面に表示できます．それぞれのパターンのフォントはパターンジェネレータ・テーブルに定義されます(**図4.27**)．パターンジェネレータ・テーブルの先頭アドレスはR#4で指定します．ただし指定できるのはアドレスの上位6ビット(A16～A11)のみです.

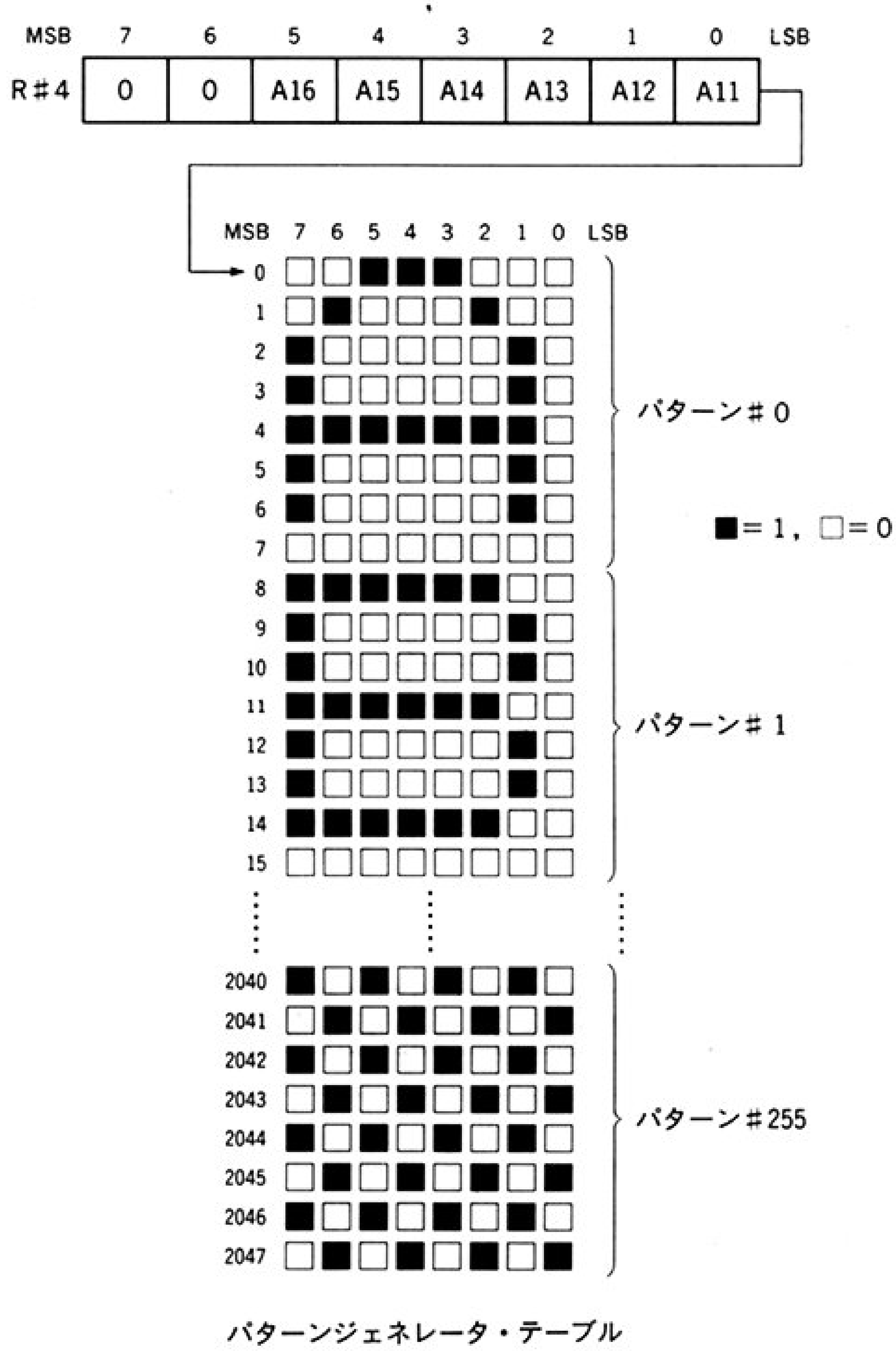


図4.27　GRAPHIC 1モードのパターンジェネレータ・テーブルの構造

● カラーテーブル

また，8パターンごとの色指定をカラーテーブルで行います．各パターンのビットが"0"の部分と"1"の部分の色を指定できます(図4.28)．カラーテーブルの先頭アドレスはR#3とR#10で指定します．ただし指定できるのはアドレスの上位11ビット(A16～A6)のみです．

● パターンネーム・テーブル

パターンネーム・テーブルは768バイトの大きさを持ち，画面上のパターンと1対1に対応しています(図4.29)．パターンネーム・テーブルの先頭アドレスはR#2で指定します．ただし指定できるのはアドレスの上位7ビット(A16～A10)のみです．

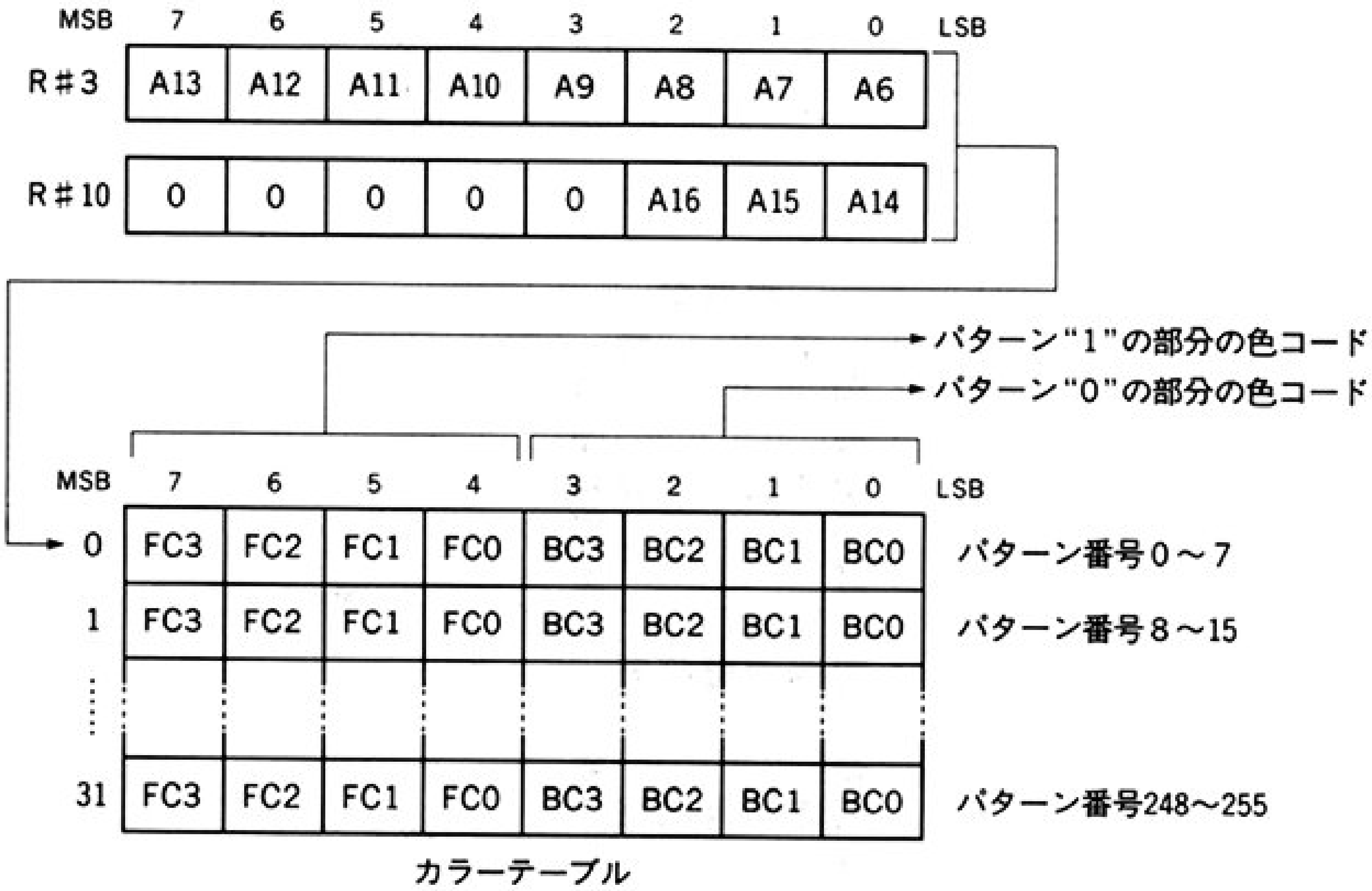


図4.28　GRAPHIC 1モードのカラーテーブルの構造

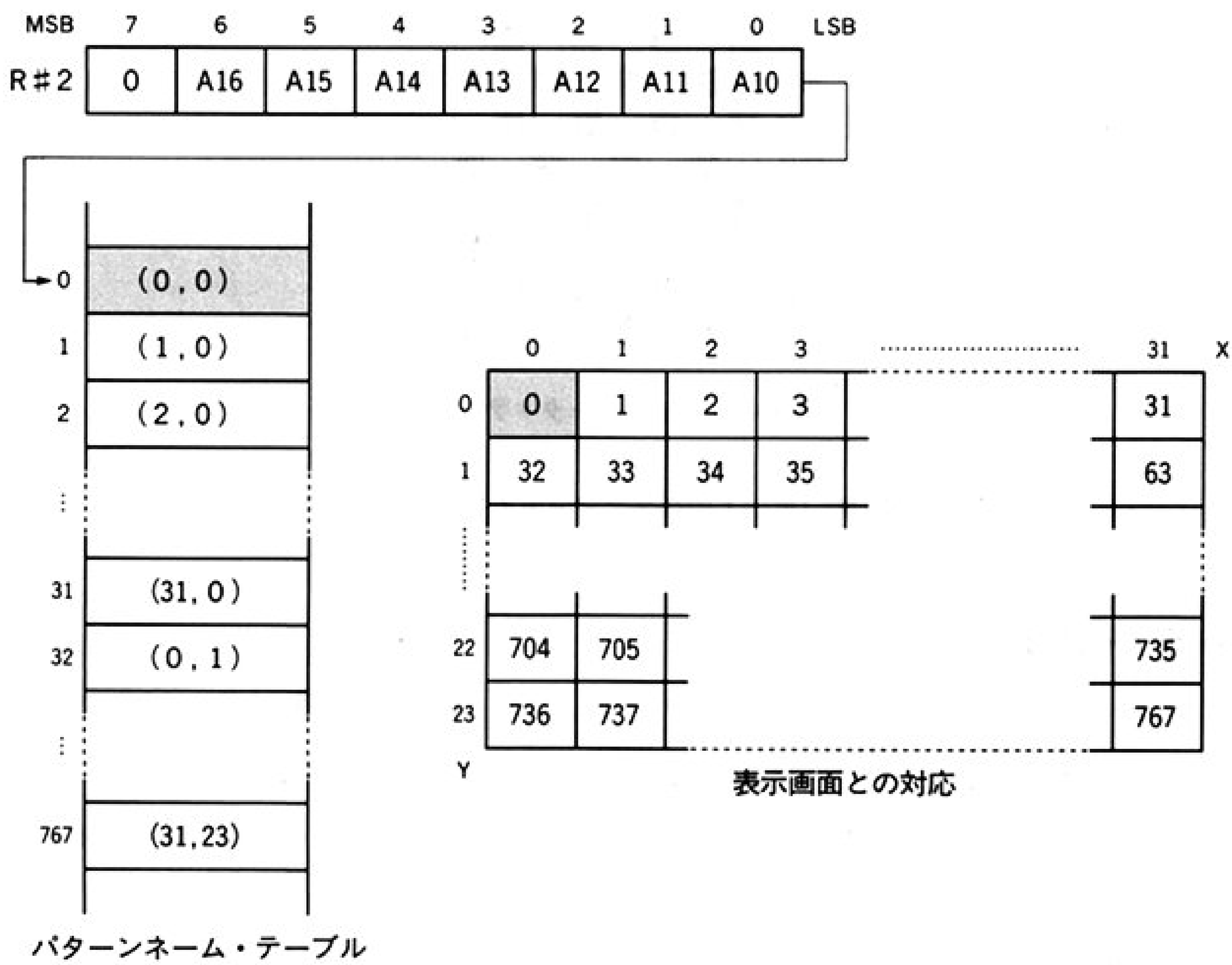


図4.29　GRAPHIC 1モードのパターンネーム・テーブルの構造

3.4.3　画面の色の指定

R#7で画面の周辺色が指定できます(図 4.30)

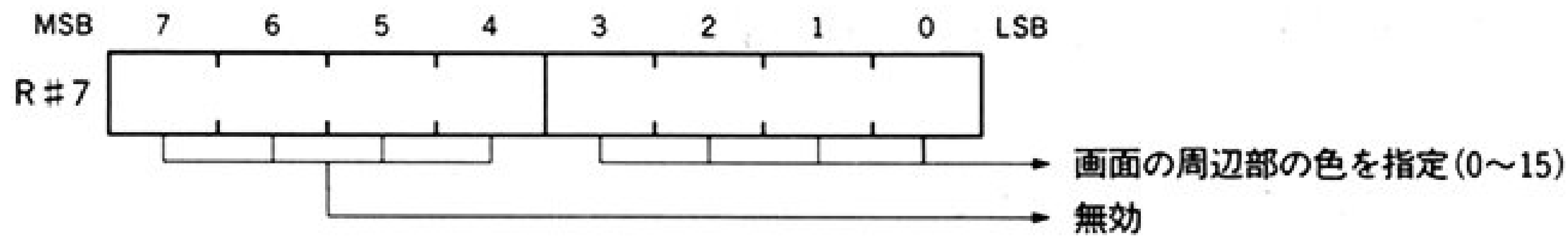


図4.30　GRAPHIC 1モードの画面の色の指定

3.5　GRAPHIC 2, 3 モード

GRAPHIC 2, 3モードとは以下に示す仕様の画面モードをいいます.

画　面	横32パターン×縦24パターン 512色中の16色が同時に表示可能
パターン	768種類のパターンが使用可能 パターンの大きさは横8ドット×縦8ドット パターンごとに任意の図形定義が可能 ただし，横8ドット内に2色までしか使用できない
必要メモリ	パターンフォント用…6144バイト カラーテーブル用……6144バイト
スプライト	GRAPHIC 2はスプライトモード1 GRAPHIC 3はスプライトモード2
BASIC	GRAPHIC 2はSCREEN 2に相当 GRAPHIC 3はSCREEN 4に相当

3.5.1　GRAPHIC 2, 3 モードの設定

GRAPHIC 2, 3 モードの設定は，それぞれ**図4.31**に示すとおりです.

GRAPHIC2モードの設定

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	0	0	1	•	
R#1	•	•	•	0	0	•	•	•	

GRAPHIC3モードの設定

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	0	1	0	•	
R#1	•	•	•	0	0	•	•	•	

図4.31　GRAPHIC 2, GRAPHIC 3モードの設定

3.5.2　GRAPHIC 2,3 モードの画面構造

● パターンジェネレータ・テーブル

このモードではGRAPHIC 1と同等のパターンジェネレータ・テーブルを3組用意し，768種類のパターンの表示が可能です．したがって画面上に重複するパターンが1つも表示されないようにすることができ，その状態でパターンジェネレータ・テーブル側を操作すれば疑似的に256×192ドットのグラフィック表示を行うことができます．パターンジェネレータ・テーブルの先頭アドレスはR#4で指定します．ただし指定できるのはアドレス上位4ビット(A16～A13)のみであるため，設定可能なアドレスは00000Hから8Kバイト単位の位置となります．(図4.32)．

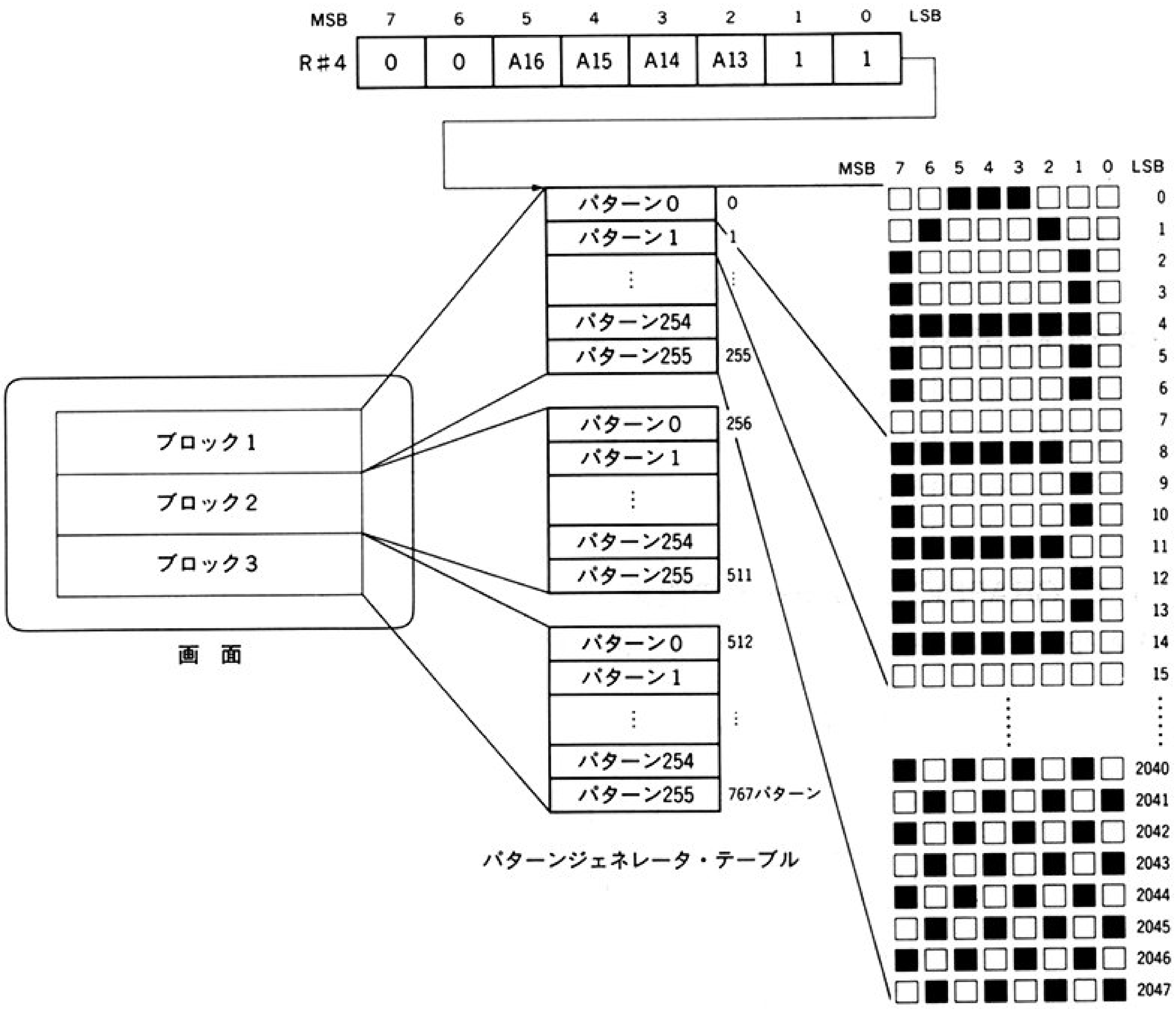


図4.32　GRAPHIC 2, GRAPHIC 3モードのパターンジェネレータ・テーブルの構造

● カラーテーブル

また，パターンジェネレータ・テーブルと同じサイズのカラーテーブルを持ち，各パターンの横1ラインごとにビットが“0”の部分と“1”の部分の色を指定できます(図4.33)．カラーテーブルの先頭アドレスはR#3とR#10で指定します．ただし指定できるのはアドレスの上位4ビット(A16～A13)のみです．

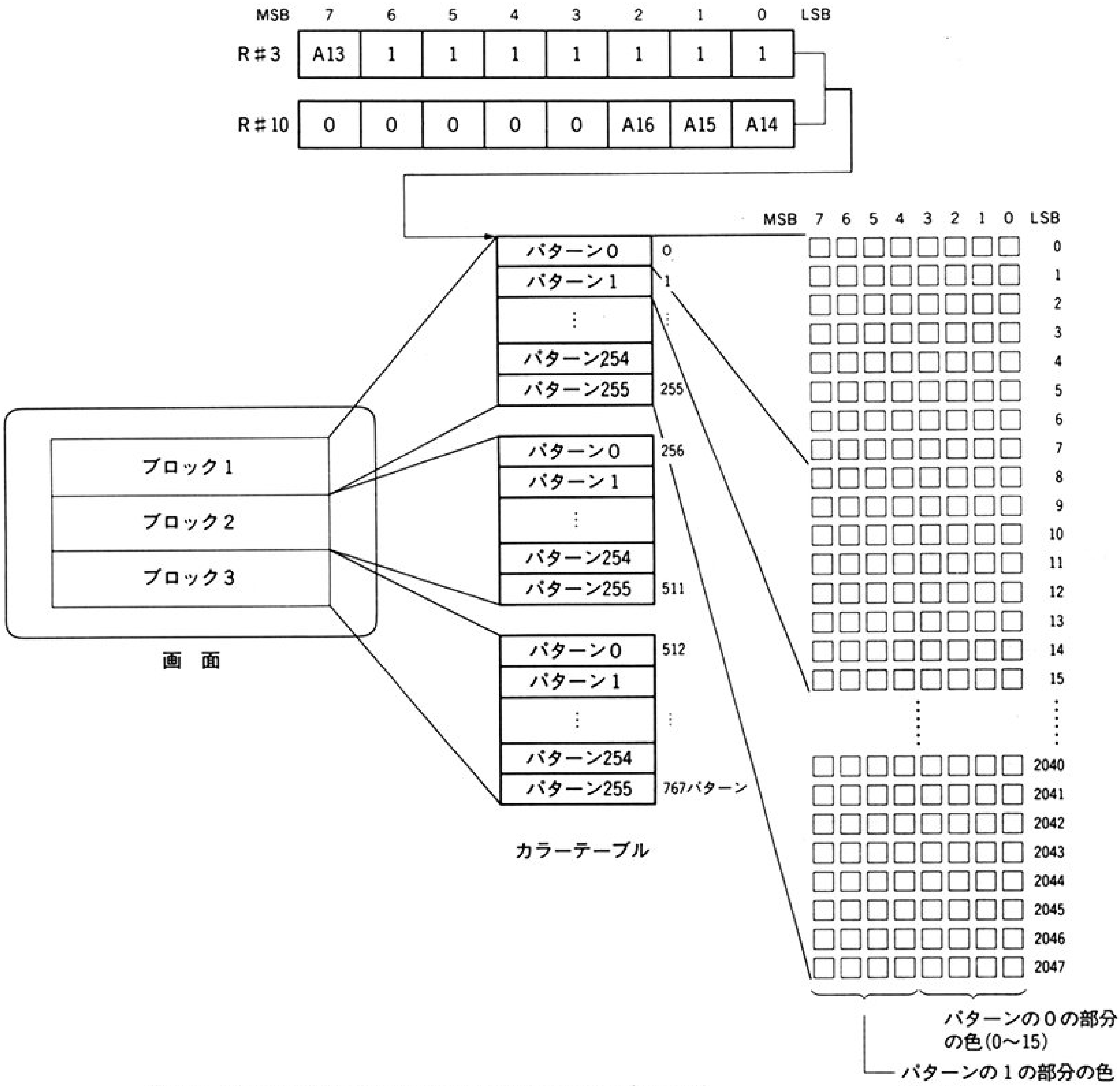
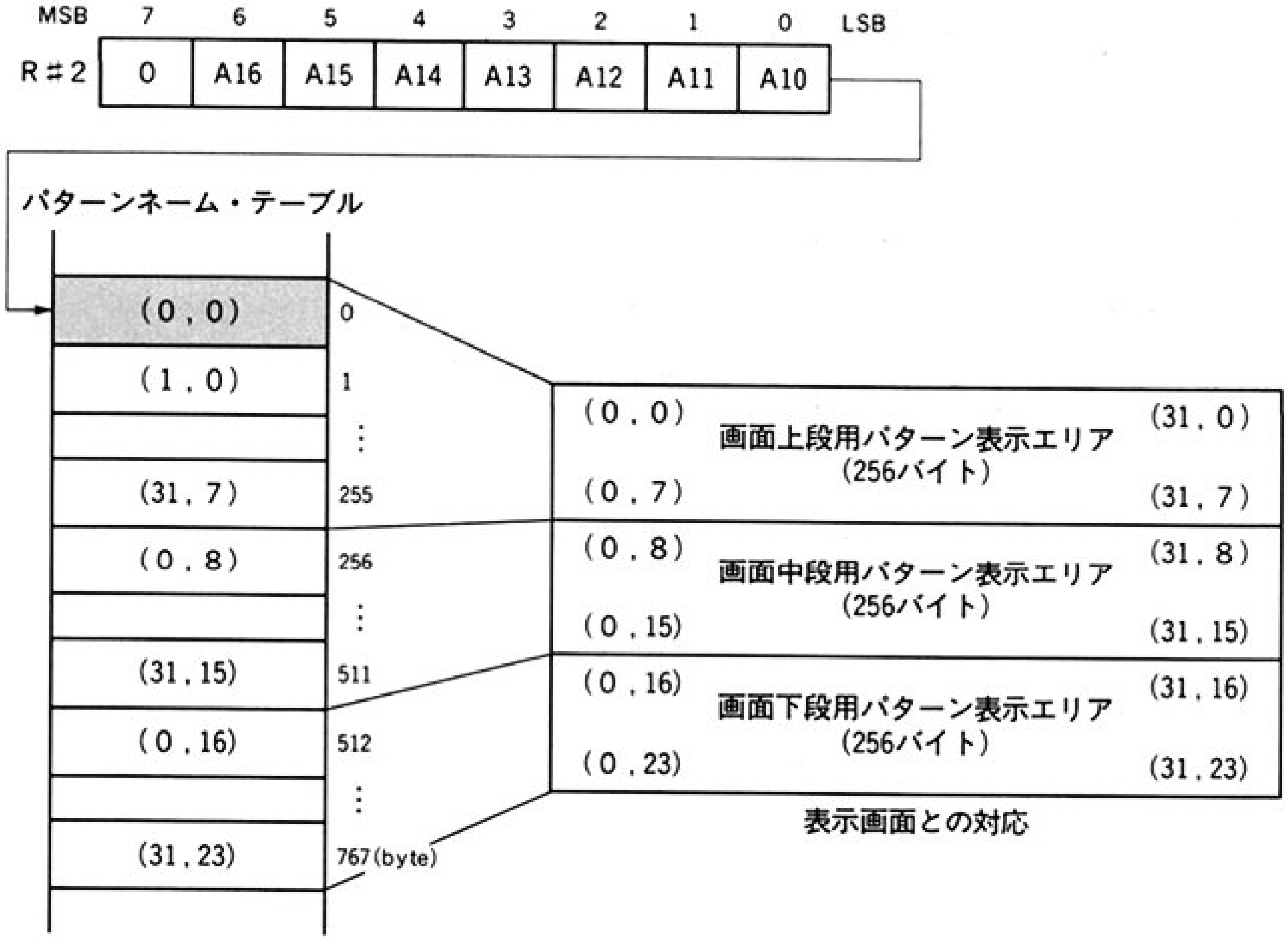


図4.33　GRAPHIC 2, GRAPHIC 3モードのカラーテーブルの構造

● パターンネーム・テーブル

パターンネーム・テーブルは上段，中段，下段の3つに分割され，それぞれがパターンジェネレータの別々の256バイトを参照してパターンを表示します(図4.34)．このような方法をとるこ

とによって，パターンネーム・テーブル上の768バイトがすべて異なったパターンフォントを表示することが可能になります．



実際にパターンネーム・テーブルに固定される内容

Y座標 ＼ X座標	0	8	16 …………	240	248　…　255
0	&H0	&H8		&HF0	&HF8
	&H1	&H9		&HF1	&HF9
	&H2	&HA		&HF2	&HFA
	&H3	&HB	…………	&HF3	&HFB
	&H4	&HC		&HF4	&HFC
	&H5	&HD		&HF5	&HFD
	&H6	&HE		&HF6	&HFE
	&H7	&HF		&HF7	&HFF
8	&H100	&H108		&H1F0	&H1F8
	&H101	&H109		&H1F1	&H1F9
	&H102	&H10A		&H1F2	&H1FA
	&H103	&H10B	………	&H1F3	&H1FB
	&H104	&H10C		&H1F4	&H1FC
	&H105	&H10D		&H1F5	&H1FD
	&H106	&H10E		&H1F6	&H1FE
	&H107	&H10F		&H1F7	&H1FF
16 ⋮	⋮	⋮	(256×192ドット)	⋮	⋮
184	&H1700	&H1708		&H17F0	&H17F8
	&H1701	&H1709		&H17F1	&H17F9
	&H1702	&H170A		&H17F2	&H17FA
	&H1703	&H170B	………	&H17F3	&H17FB
	&H1704	&H170C		&H17F4	&H17FC
	&H1705	&H170D		&H17F5	&H17FD
	&H1706	&H170E		&H17F6	&H17FE
191	&H1707	&H170F		&H17F7	&H17FF

注)値はパターンジェネレータ・テーブルのベースアドレスからのオフセットです

図4.34　GRAPHIC 2, GRAPHIC 3モードのパターンネーム・テーブルの構造

3.5.3　画面の色の指定

R#7で画面の周辺色が指定できます(図4.35).

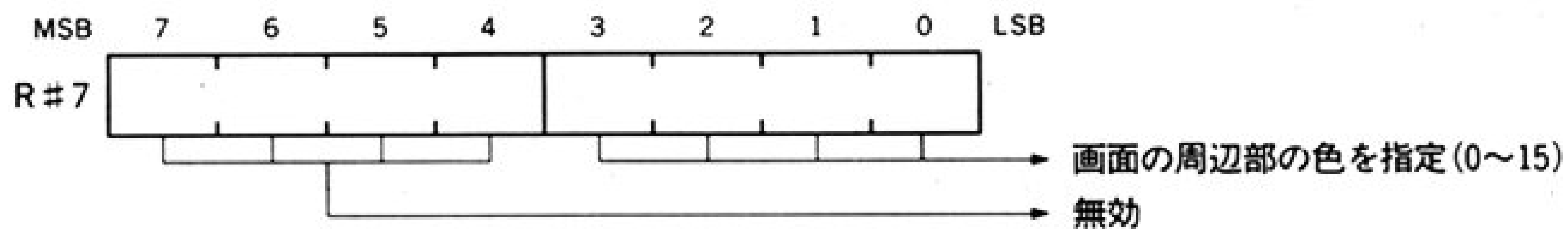


図4.35　GRAPHIC 2, GRAPHIC 3モードの画面の色の指定

3.6　GRAPHIC 4 モード

GRAPHIC 4 モードとは，以下に示すモードのことをいいます.

画　面：横256ドット×縦212ドット(または縦192ドット)
同時に16色が表示可能
16色のそれぞれは，512色中から任意の1色を選択可能

コマンド：VDP コマンドによる高速グラフィック可能

スプライト：モード2のスプライト機能が使用可能

必要メモリ：192ドット設定時
ビットマップ画面……24Kバイト(6000Hバイト)
(4ビット×256×192)
212ドット設定時
ビットマップ画面……26.5Kバイト(6A00Hバイト)
(4ビット×256×212)

BASIC：SCREEN 5に対応

3.6.1　GRAPHIC 4 モードの設定

GRAPHIC 4 モードの設定は図4.36に示すとおりです.

	MSB 7	6	5	4	3	2	1	0 LSB
R#0	•	•	•	•	0	1	1	•
R#1	•	•	•	0	0	•	•	•

図4.36　GRAPHIC 4モードの設定

3.6.2 GRAPHIC 4 モードの画面構造

● パターンネーム・テーブル

GRAPHIC 4 モードでは，パターンネーム・テーブルの 1 バイトが画面上の 2 ドットに対応しています．それぞれのドットの色情報は 4 ビットで表されており，16色の指定ができます（図 4.37）．パターンネーム・テーブルの先頭アドレスの指定には，R#2を用います．指定できるのはアドレスの上位 2 ビット(A16～A15)のみであり，下位15ビットはすべて“0”とみなされます．したがって，パターンネーム・テーブルが設定可能なアドレスは，00000H，08000H，10000H，18000Hの 4 箇所だけです．

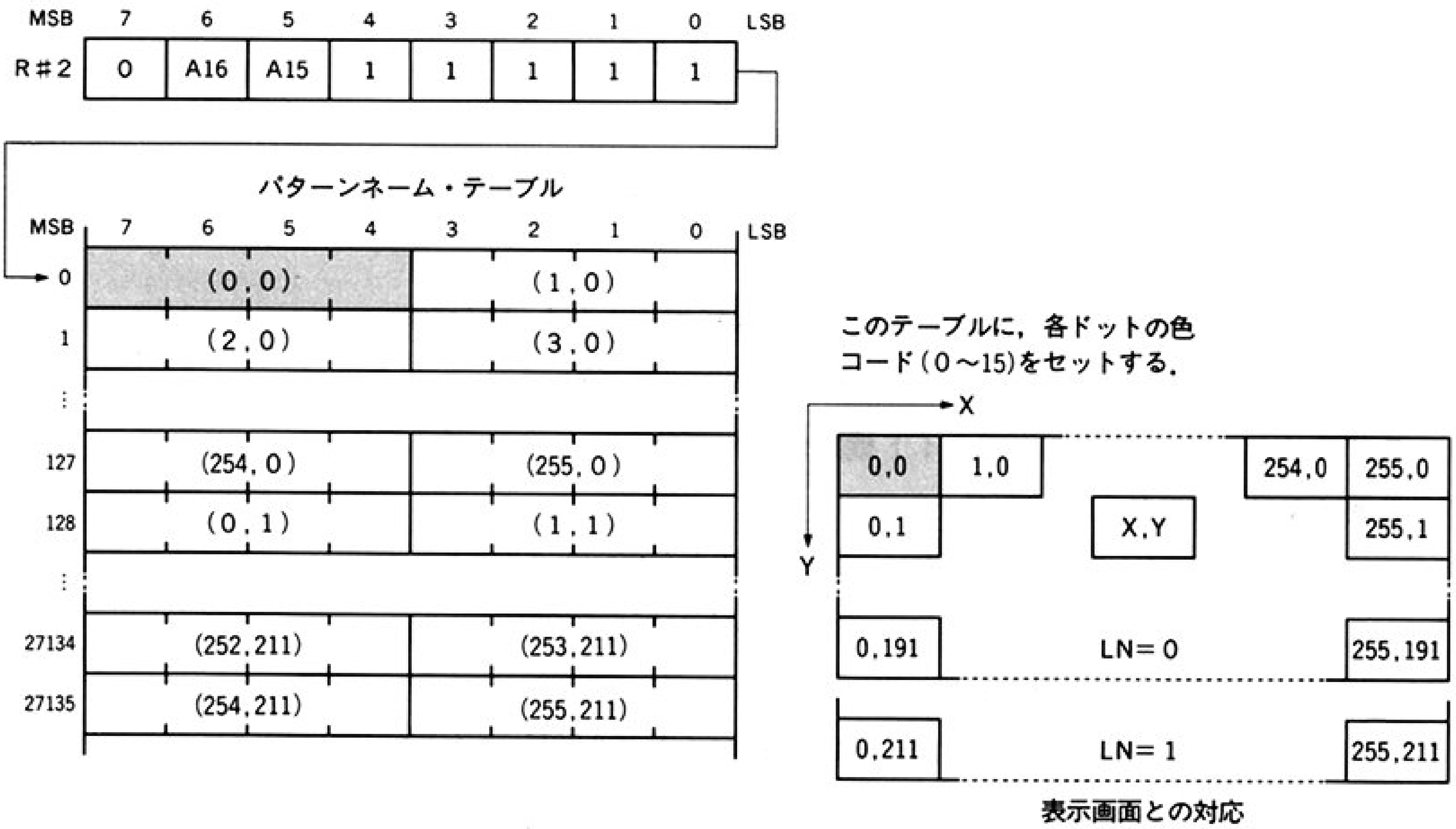


図4.37 GRAPHIC 4モードのパタールネーム・テーブルの構造

このことから，画面上で(X,Y)の座標にあるドットは，**式 4.1** でアクセスできることがわかります．**リスト 4.2** は，**式 4.1** を確認するためのプログラムです．

ADR=X/2+Y＊128+ベースアドレス

※Xが偶数のときは上位4ビット
Xが奇数のときは下位4ビットがそのドットの色を表す

式4.1 座標(X,Y)のドットをアクセスする式

リスト4.2　BASICで書いたGRAPHIC 4モード用PSET

```
100 '============================================
110 '  LIST 4.2   dot accsess of GRAPHIC 4 mode
120 '============================================
130 '
140 SCREEN 5
150 BA=0
160 FOR I=0 TO 255
170   X=I:Y=I¥2
180   COL=15
190   GOSUB 1000
200 NEXT
210 END
220 '
1000 '========================================
1010 '  PSET (X,Y),COL
1020 '   COL:color  BA:graphic Base Address
1030 '========================================
1040 '
1050 ADR=X¥2+Y*128+BA
1060 IF X AND 1 THEN BIT=&HF0:C=COL ELSE BIT=&HF:C=COL*16
1070 D=VPEEK(ADR)
1080 D=(D AND BIT) OR C
1090 VPOKE ADR,D
1100 RETURN
```

3.6.3　画面の色の指定

R#7で画面の周辺色が指定できます(図4.38).

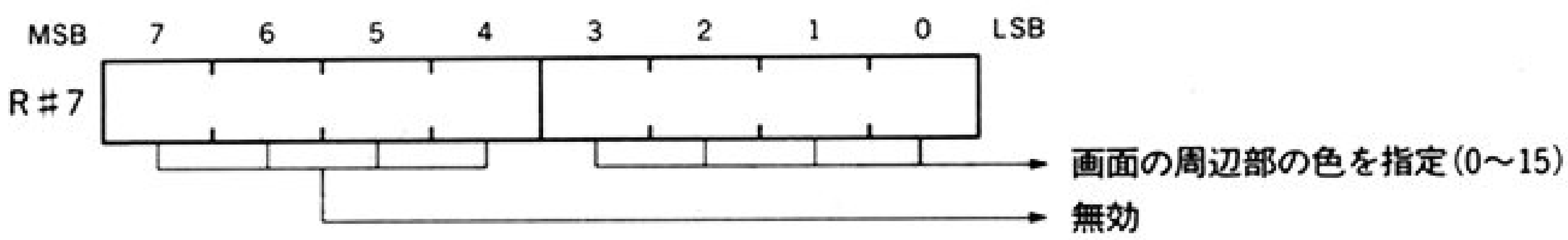


図4.38　GRAPHIC 4モードの画面の色の指定

3.7 GRAPHIC 5 モード

GRAPHIC 5 モードとは，以下に示す仕様の画面モードをいいます．

画　面：	横512ドット×縦212ドット（または縦192ドット） 同時に４色が表示可能 ４色のそれぞれは，512色中から任意の１色を選択可能
コマンド：	ハードウェアによるグラフィックコマンドが使用可能
スプライト：	モード２のスプライト機能が使用可能
必要メモリ：	縦192ドット設定時 ビットマップ画面……24Kバイト（6000Hバイト） （２ビット×512×192） 縦212ドット設定時 ビットマップ画面……26.5Kバイト（6A00Hバイト） （２ビット×512×212）
BASIC：	SCREEN 6に対応

3.7.1 GRAPHIC 5 モードの設定

GRAPHIC 5 モードへの設定は図4.39に示すとおりです．

	MSB 7	6	5	4	3	2	1	0 LSB
R#0	•	•	•	•	1	0	0	•
R#1	•	•	•	0	0	•	•	•

図4.39　GRAPHIC 5モードの設定

3.7.2 パターンネーム・テーブル

GRAPHIC 5 モードでは，パターンネーム・テーブルの１バイトが画面上の４ドットに対応しています．それぞれのドットの色情報は２ビットで表され，４色が指定できます．パターンネーム・テーブルの設定は，GRAPHIC5 モードと同様，R#2にアドレスの上位２ビットを書き込むことによって行います．設定可能なアドレスは，00000H，08000H，10000H，18000Hの４箇所です（図4.40）．

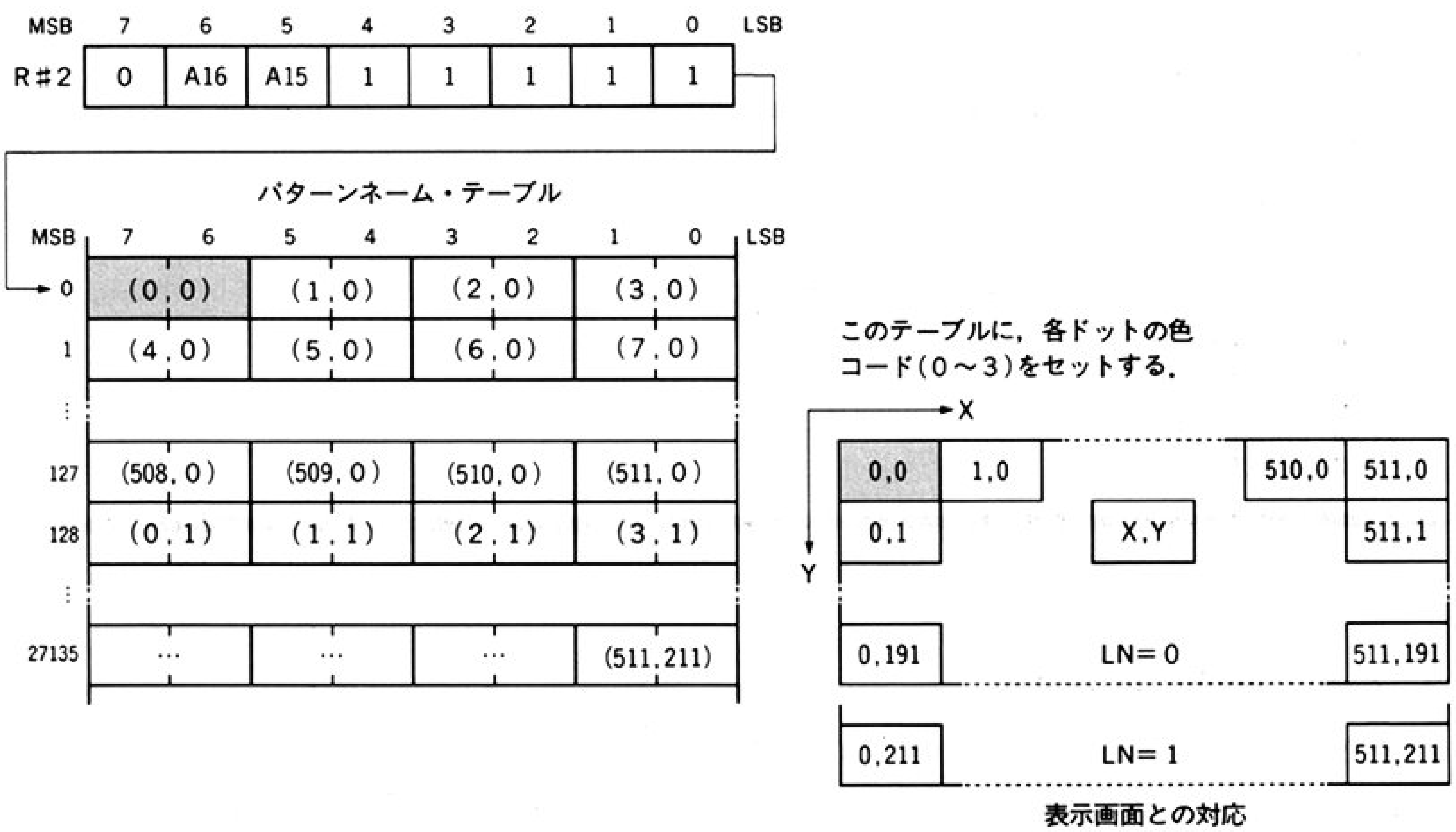


図4.40 GRAPHIC 5モードのパターンネーム・テーブルの構造

このことから，(X,Y)の位置にあるドットは，**式4.2**でアクセスできることになります．**リスト4.3**がその確認のためのプログラムです．

ADR=X／4+Y＊128+ベースアドレス

※X MOD 4が0のときは7，6ビット
X MOD 4が1のときは5，4ビット
X MOD 4が2のときは3，2ビット
X MOD 4が3のときは1，0ビットがそのドットの色を表す

式4.2 座標(X,Y)のドットをアクセスする式

リスト4.3 BASICで書いたGRAPHIC 5モード用PSET

```
100 '==========================================
110 '  LIST 4.3   dot accsess of GRAPHIC 5 mode
120 '==========================================
130 '
140  SCREEN 6
150  BA=0
160  FOR I=0 TO 511
170    X=I:Y=I¥2
180    COL=3
190    GOSUB 1000
200  NEXT
210  END
220 '
```

```
1000 '==========================================
1010 '  PSET(X,Y)
1020 '    COL:color  BA:graphic Base Address
1030 '==========================================
1040 '
1050 ADR=X¥4+Y*128+BA
1060 LP=X MOD 4
1070 IF LP=0 THEN BIT=&H3F:C=COL*&H40
1080 IF LP=1 THEN BIT=&HCF:C=COL*&H10
1090 IF LP=2 THEN BIT=&HF3:C=COL*&H4
1100 IF LP=3 THEN BIT=&HFC:C=COL
1110 D=VPEEK(ADR)
1120 D=(D AND BIT) OR C
1130 VPOKE ADR,D
1140 RETURN
```

3.7.3　画面の色の設定

GRAPHIC 5 モードでは画面周辺色とスプライトについてハードウェア的にタイリングが行われます．これらの色の指定は他のモードと同じように 4 ビットで行われますが，4 ビットのうち上位 2 ビットはX座標が偶数のドット，下位 2 ビットはX座標が奇数のドットの色をそれぞれ表します(**図 4.41**)．

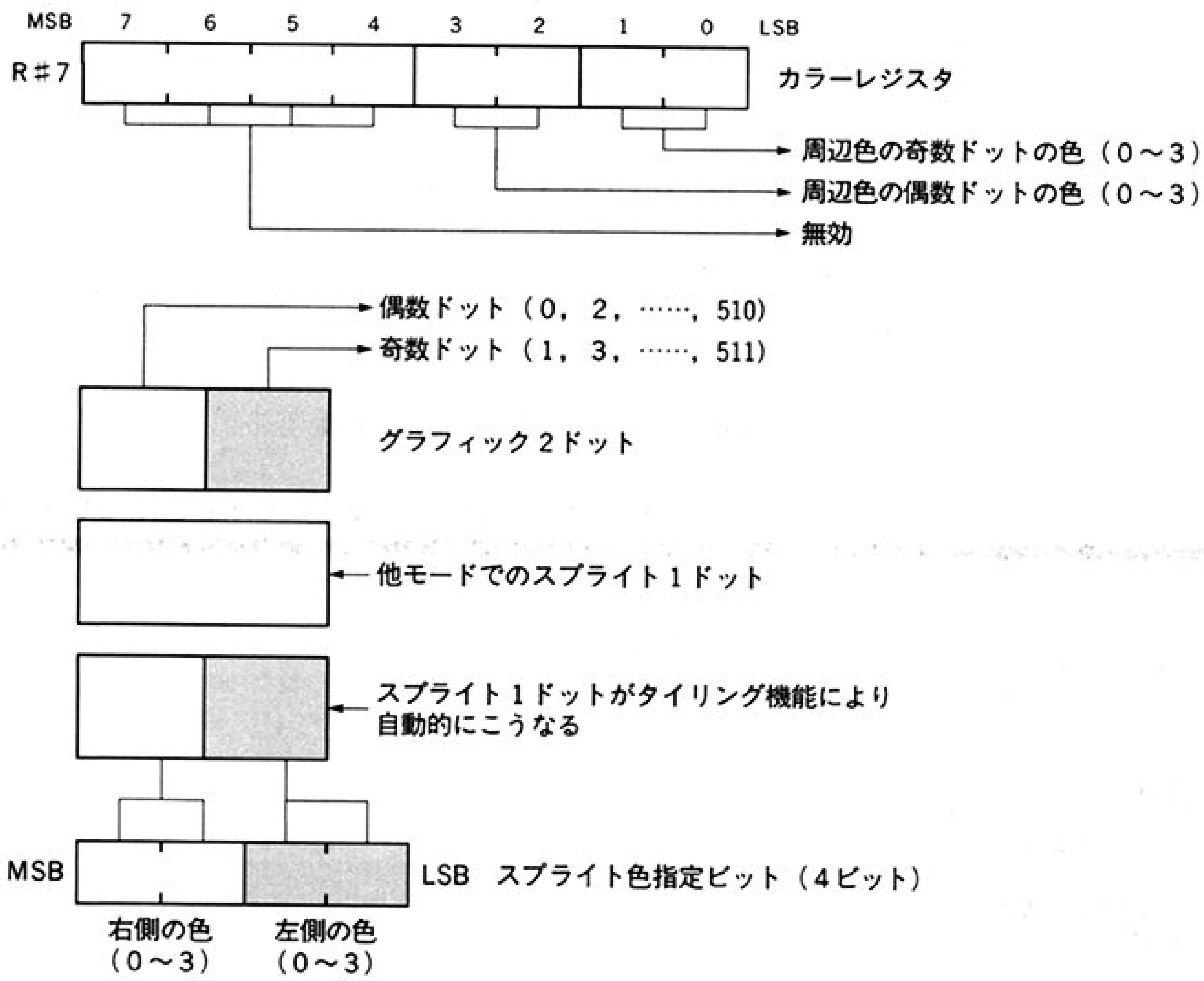


図4.41　GRAPHIC 5モードの画面の色の指定

3.8　GRAPHIC 6 モード

GRAPHIC 6 モードとは，以下に示す仕様の画面モードをいいます．

画　面：横512ドット×縦212ドット（または縦192ドット）
16色が同時に表示可能
16色のそれぞれは，512色中から任意の1色を選択可能
コマンド：ハードウェアによるグラフィックコマンドが使用可能
スプライト：モード2のスプライト機能が使用可能
必要メモリ：縦192ドット設定時
ビットマップ画面用……48Kバイト(C000Hバイト)
(4ビット×512×192)
縦212ドット設定時
ビットマップ画面用……53Kバイト(D400Hバイト)
(4ビット×512×212)
ただし，ハードウェアの都合上VRAMが64Kバイトの機種ではまったくこのモードは使用不可能
BASIC：SCREEN 7に対応

3.8.1　GRAPHIC 6 モードの設定

GRAPHIC 6 モードへの設定は図4.42に示すとおりです．

	MSB 7	6	5	4	3	2	1	0 LSB
R#0	•	•	•	•	1	0	1	•
R#1	•	•	•	0	0	•	•	•

図4.42　GRAPHIC 6モードの設定

3.8.2　パターンネーム・テーブル

GRAPHIC 6 モードは，パターンネーム・テーブルの1バイトが画面上の2ドットに対応しています．それぞれのドットの色情報は4ビットで表されており，16色の指定ができます(図4.43)．パターンネーム・テーブルの先頭アドレス指定は，R#2にアドレスの最上位ビットを書き込むことによって行います．設定可能なパターンネーム・テーブルのアドレスは00000Hまたは10000

Hの2箇所だけです．座標(X,Y)のドットは，**式4.3**でアクセスすることができます．**リスト4.4**がその確認のためのプログラムです．

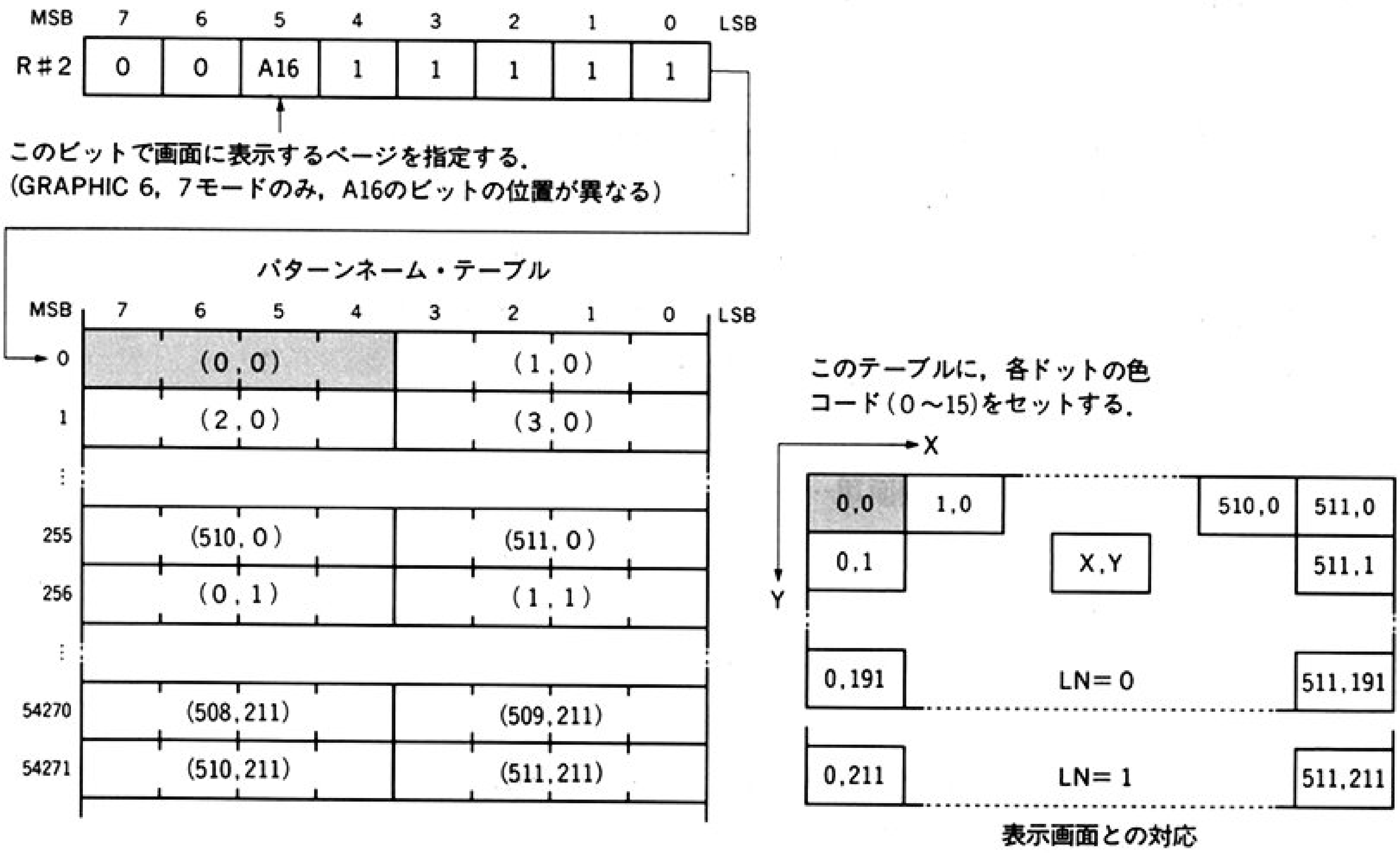


図4.43　GRAPHIC 6モードのパターンネーム・テーブルの構造

ADR＝X／2＋Y＊256＋ベースアドレス

※Xが偶数のときは上位4ビット
　Xが奇数のときは下位4ビットがそのドットの色を表す

式4.3　座標(X,Y)のドットをアクセスする式

リスト4.4　BASICで書いたGRAPHIC 6モード用PSET

```
100 '==========================================
110 '  LIST 4.4   dot accsess of GRAPHIC 6 mode
120 '==========================================
130 '
140 SCREEN 7
150 BA=0
160 FOR I=0 TO 511
170   X=I: Y=I¥2: COL=15: GOSUB 1000
180 NEXT
190 END
200 '
1000 '==========================================
1010 '  PSET (X,Y)
1020 '    COL:color  BA:graphic Base Address
1030 '==========================================
1040 '
1050 ADR=X¥2+Y*256+BA
1060 IF X AND 1 THEN BIT=&HF: C=COL ELSE BIT=&HF0 :C=COL*16
1070 VPOKE ADR,(VPEEK(ADR) AND BIT) OR COL
1080 RETURN
```

3.8.3 画面の色の指定

R#7で画面の周辺色の指定ができます(図 4.44).

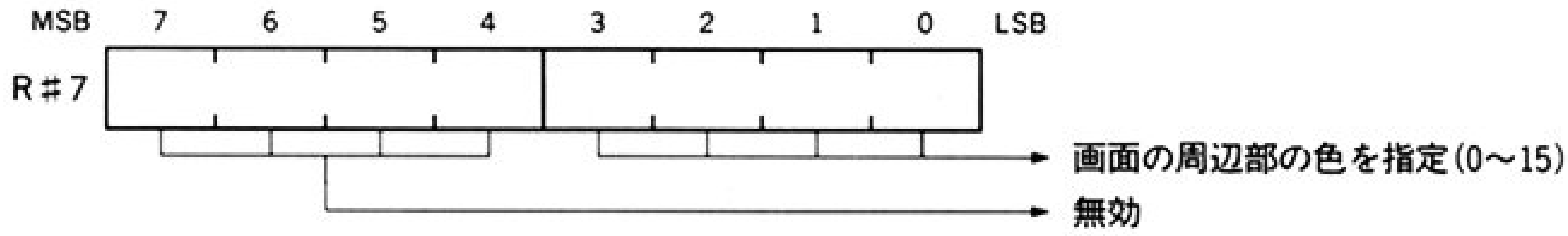


図4.44 GRAPHIC 6モードの画面の色の指定

3.9 GRAPHIC 7 モードの使用法

GRAPHIC.7 モードとは，以下に示す仕様の画面モードをいいます.

画　面：横256ドット×縦212ドット(または縦192ドット)
256色が同時に表示可能

コマンド：ハードウェアによるグラフィックコマンドが使用可能

スプライト：モード 2 のスプライト機能が使用可能

必要メモリ：縦192ドット設定時
ビットマップ画面用……48Kバイト(C000Hバイト)
(8ビット×256×192)
縦212ドット設定時
ビットマップ画面用……53Kバイト(D400Hバイト)
(8ビット×256×212)
GRAPHIC 6 と同様に VRAM が 64Kバイトの機種ではまったくこのモードは使用不可能

BASIC：SCREEN 8 に対応

3.9.1 GRAPHIC 7 モードの設定

GRAPHIC 7 モードの設定は，以下に示すとおりです.

MSB	7	6	5	4	3	2	1	0	LSB
R#0	•	•	•	•	1	1	1	•	
R#1	•	•	•	0	0	•	•	•	

図4.45 GRAPHIC 7モードの設定

3.9.2 パターンネーム・テーブル

GRAPHIC 7 モードはすべてのモードの中で最も単純な構成であり，画面上の 1 ドットがそのままパターンネーム・テーブルの 1 バイトに対応します．また，テーブルに書き込んだ 1 バイトの値はそのまま256通りの色を表します．パターンネーム・テーブルの先頭アドレス指定は，GRAPHIC 6 モードの場合と同様，R#2で行います．設定可能なパターンネーム・テーブルのアドレスは00000Hまたは10000Hの 2 箇所だけです(**図 4.46**)．

1 バイトのデータは**図 4.47** のように，緑 3 ビット，赤 3 ビット，青 2 ビットの輝度情報を直接表しています．画面上で(X, Y)の座標にあるドットは，**式 4.4** でアクセスすることができます．

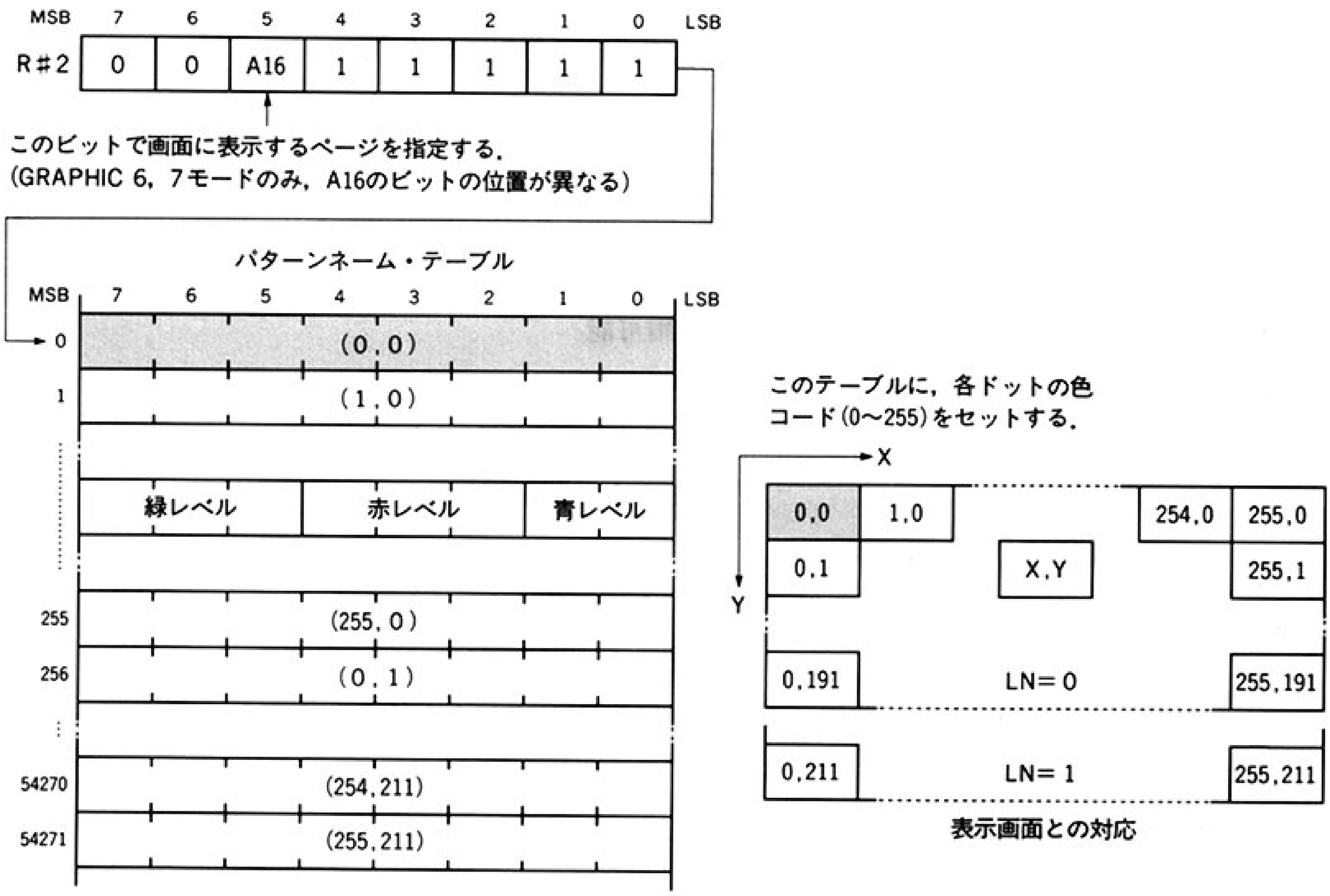


図4.46　GRAPHIC 7モードのパターンネーム・テーブルの構造

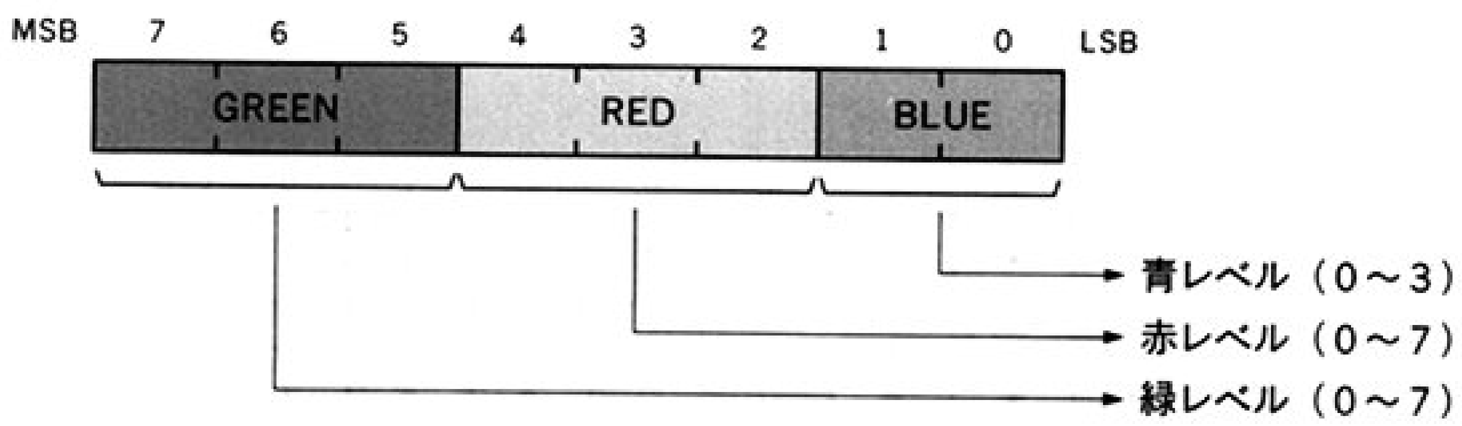


図4.47　RGBの輝度情報

ADR=X+Y＊256+ベースアドレス

式4.4　座標(X, Y)のドットをアクセスする式

3.9.3　画面の色の指定

R#7で画面の周辺色を指定できます(図 4.48)

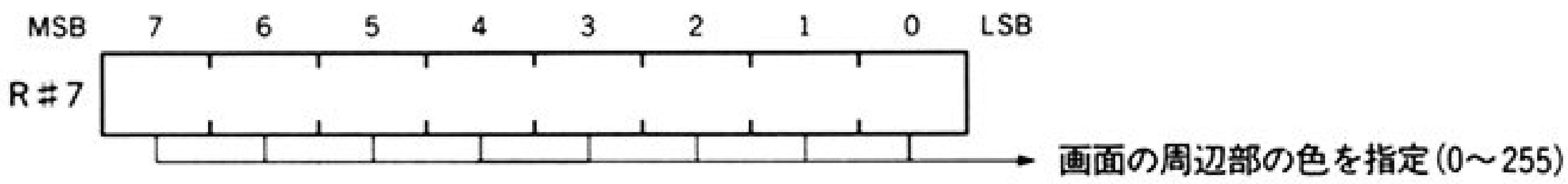


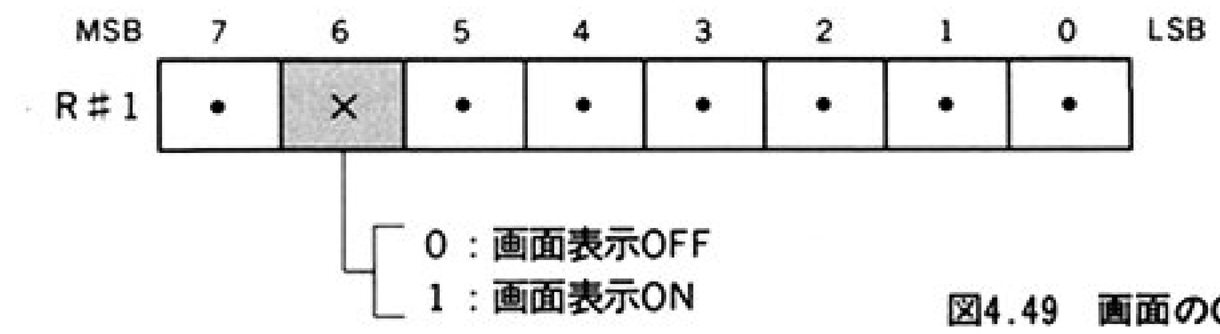
図4.48　GRAPHIC 7モードの画面の色の指定

4章 画面表示に関する諸機能

MSX-VIDEOは画面のON/OFFや表示位置の補正など，画面表示に関する種々の細かい設定が可能です．本章では，それらの設定によって使用可能なMSX-VIDEOの機能を説明します．

● 画面のON／OFF

R#1のビット6を用いて，画面のON/OFFを行います(図4.49)．OFFにすると，画面はR#7レジスタの下位4ビット(GRAPHIC 7の場合は8ビット)で指定される色に変わり，その際VDPコマンドで描画を行う際の速度が少し向上します．



BASICによる実行例：

```
VDP(1)=VDP(1) AND &B10111111 ←画面表示OFF
VDP(1)=VDP(1) OR &B01000000 ←画面表示ON
```

図4.49 画面のON／OFF

● 画面表示位置の補正

R#18は画面の表示位置を補正するためのレジスタです(図4.50)．BASICの“SET ADJUST”命令の機能に相当します．

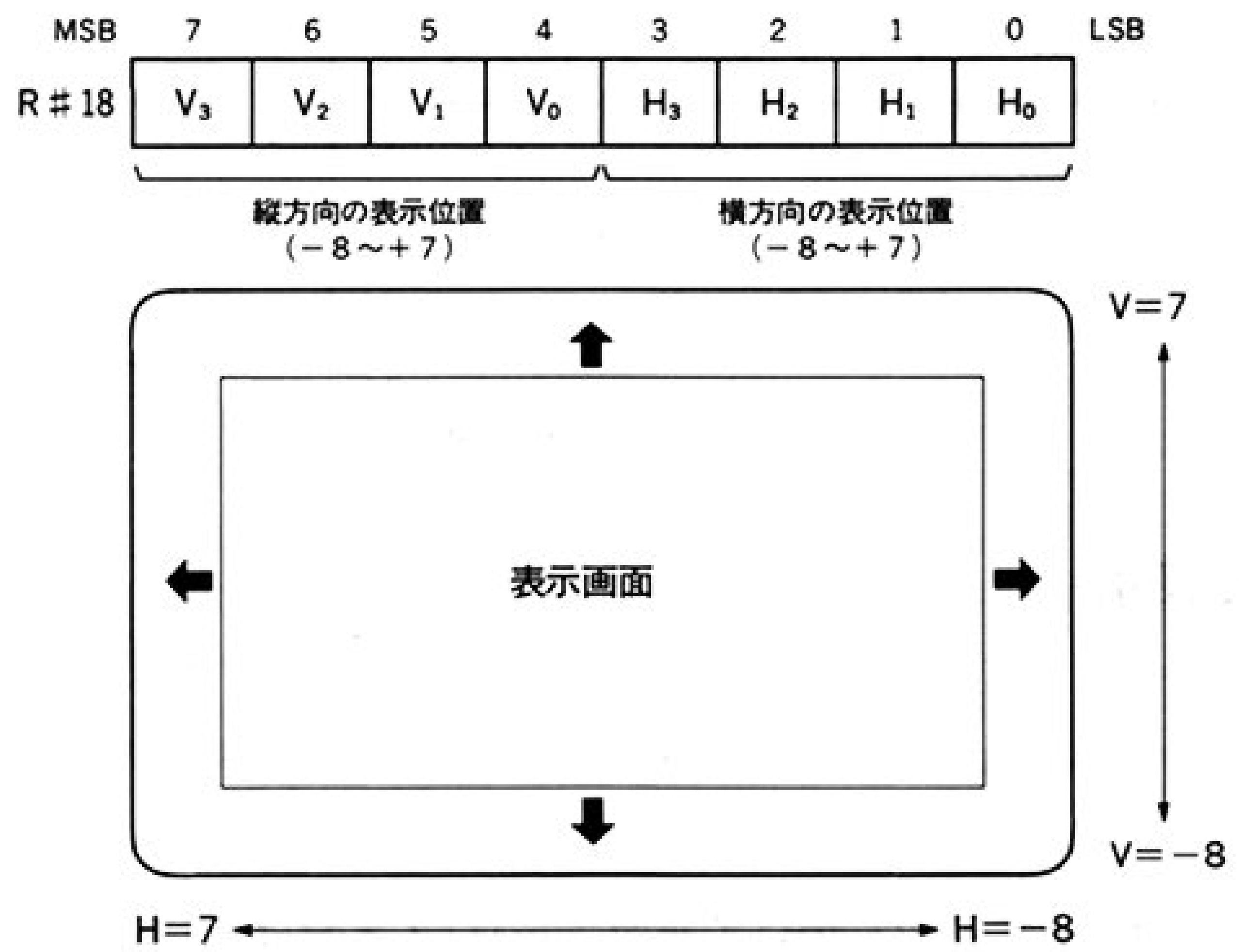


図4.50 表示位置の補正

● Y方向ドット数の切り換え

R#9のビット7を用いて，画面のY方向のドット数を192ドットまたは212ドットのどちらかに切り換えることができますが，この機能は，TEXT 2 およびGRAPHIC 4～7の5種類の画面モードにおいてのみ有効です．なお，TEXT 2 モードで212ドットを設定した場合，テキストの行数は26.5(＝212/8)となって，27行目の文字は上半分しか表示されません．

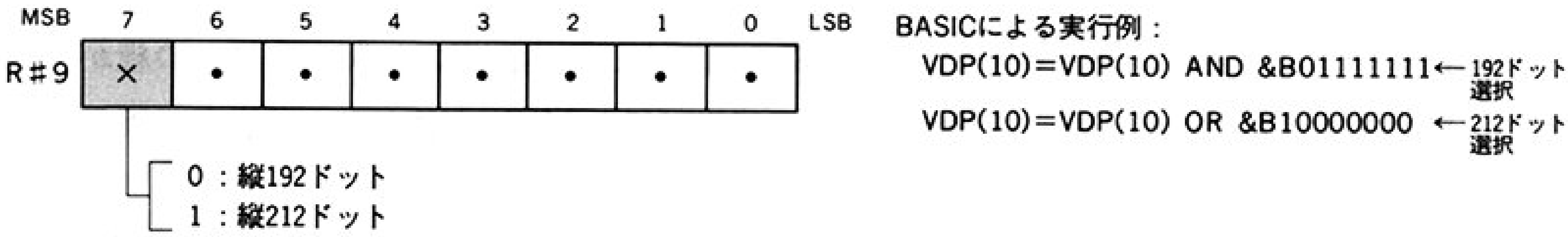


図4.51　縦のドット数切り換え

● 表示ページの切り換え

GRAPHIC 4～7 モードでは，R#2でパターンネーム・テーブルの先頭アドレスを設定することにより，簡単に表示ページが切り換えられます．実際，BASICの“SET PAGE”命令の第2パラメータでは，このようにして表示ページを切り換えています．

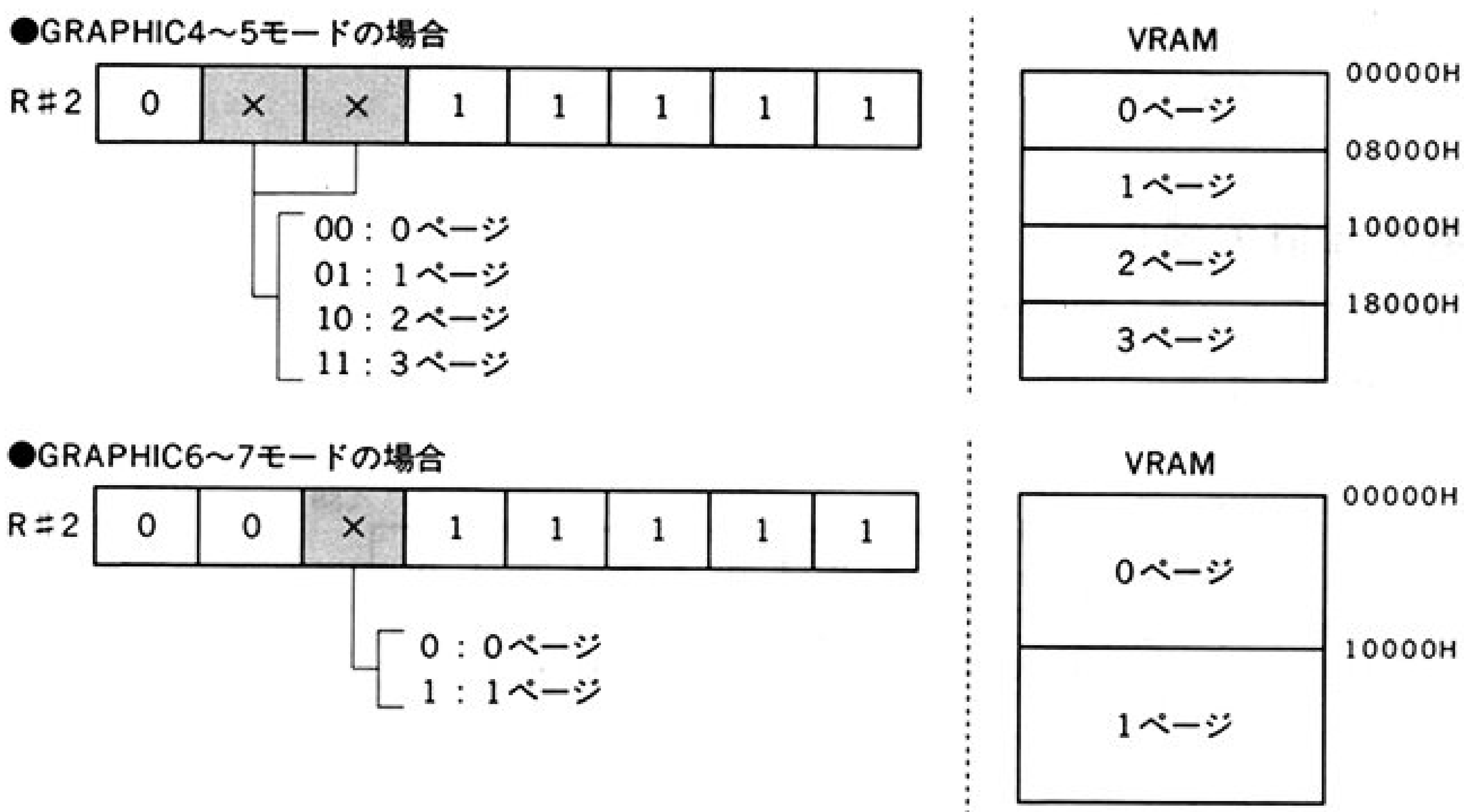


図4.52　ページ切り換え

● 画面の自動交互表示

GRAPHIC 4～7のモードでは，以下の方法を用いて2枚のページを交互に表示することができます．表示させられるページは，0ページと1ページ，または2ページと3ページの組み合わせです．

交互表示を開始するには，まずR#2を用いて奇数ページ(1または3)を選択し，さらにR#13に

画面交代の周期を設定します．R#13の上位4ビットは偶数ページの表示時間，下位4ビットは奇数ページの表示時間を，それぞれ1/6秒単位で表したものです．両方の時間ともに0を設定した場合には，奇数ページのみ表示されます．

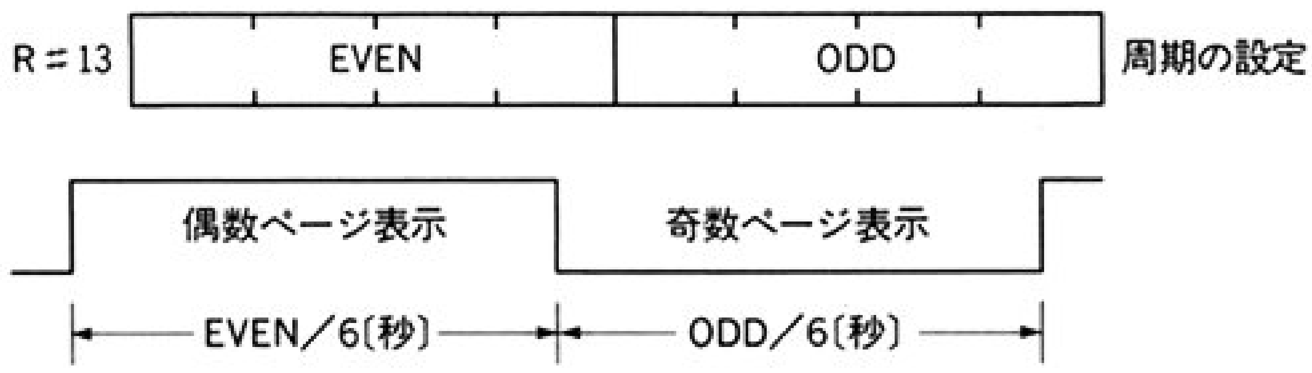


図4.53　画面交代の周期の設定

● インターレースモードの設定

インターレースとは，通常の画面と走査線を通常より半ライン下にずらした画面を高速に切り換えながら表示し，疑似的にY方向の分解能を2倍に高める手法です．MSX-VIDEOでは，R#9のビット3を"1"にすることによって，インターレースモードが設定されます．2枚の画面は1/60秒ごとに切り換わります．

また，GRAPHIC 4～7の画面モードで奇数ページを選択し，さきほど述べた画面の交互表示を行う場合，普通は1/6秒を単位とするゆっくりとしたスピードで画面が切り換わるのですが，R#9のビット2を"1"としておくと，1/60秒間隔で交互表示が行われます．この機能とインターレース機能を組み合わせると，画面の縦方向の表示ドット数を2倍にすることができます．

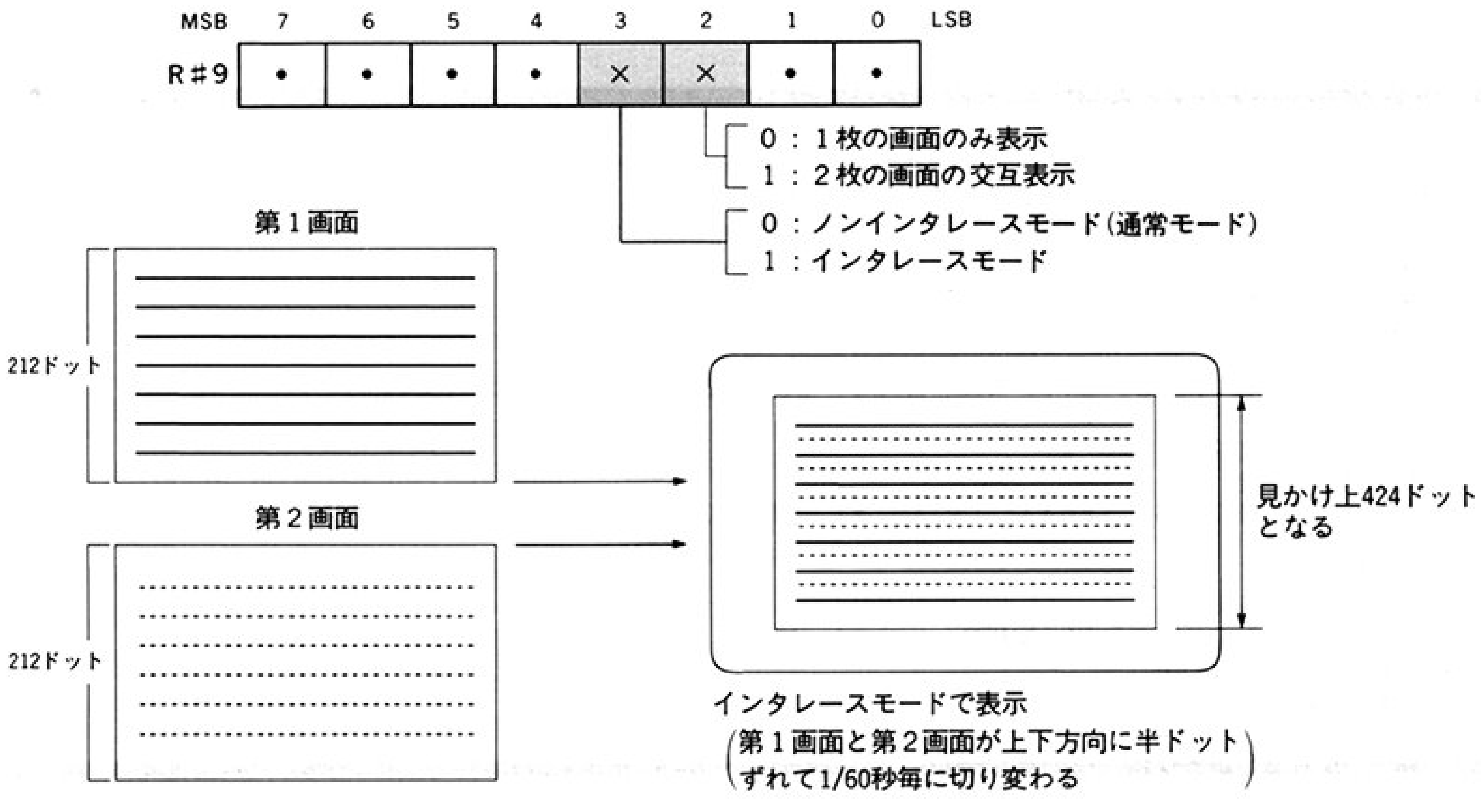


図4.54　インターレースモードの設定

リスト4.5　インターレースの使用例

```
1000 '==================================================
1010 '  List 4.5  interlace mode
1020 '==================================================
1030 '
1040 COLOR 15,0,0 : SCREEN 5,,,,,0     'non interlace mode
1050 '
1060 SET PAGE 0,0 : CLS
1070 LINE (32,0)-(64,120),15,BF
1080 SET PAGE 1,1 : CLS
1090 LINE(192,91)-(224,211),15,BF
1100 '
1110 VDP(10)=VDP(10) OR &B00001100     'interlace mode!!!
1120 '
1130 FOR I=32 TO 192
1140   SET PAGE 1,0
1150   LINE(I,0)-STEP(0,120),0
1160   LINE(I+33,0)-STEP(0,120),15
1170   SET PAGE 1,1
1180   LINE(256-I,91)-STEP(0,120),0
1190   LINE(223-I,91)-STEP(0,120),15
1200 NEXT I
1210 '
1220 VDP(10)=VDP(10) AND &B11110011     'interlace off
```

● 画面の縦方向スクロール

R#23は画面の表示開始ラインを設定するレジスタです．このレジスタの値を変えることによって，画面の縦方向スクロールを行うことができます．ただし，スクロールは256ライン単位で行われますから，スプライト用テーブルなどは他のページに移動させなくてはなりません．この例を**リスト 4.6** に示します．

リスト4.6　画面スクロールの例

```
1000 '============================================================
1010 '  List 4.6  Hardware Scroll
1020 '============================================================
1030 '
1040 SCREEN 5,2: COLOR 15,0,0: CLS
1050 COPY (0,0)-(255,43) TO (0,212),,PSET  'erase (212,0)-(255,255)
1060 '
1070 FOR I=1 TO 8: D(I)=VAL(MID$("00022220",I,1))-1: NEXT
1080 '
1090 OPEN "GRP:" AS #1
1100 FOR I=0 TO 3
1110   PRESET (64,I*64): PRINT #1,"Hit CURSOR Key"
1120 NEXT
1130 '
1140 J=STICK(0)
1150   P=(P+D(J)) AND &HFF
1160   VDP(24)=P
1170 GOTO 1140
```

● 色コード0の機能指定

16種類の色コードの中で，0だけは特別な機能を持ち，この色を指定した部分は，カラーパレットP#0にどんな色が設定されていても“透明色(画面の周辺色が透けて見える)”とみなされます．しかし，R#8のビット5を“1”にするとこの機能は解消され，色コード0はパレットP#0が定義する色に変わります．

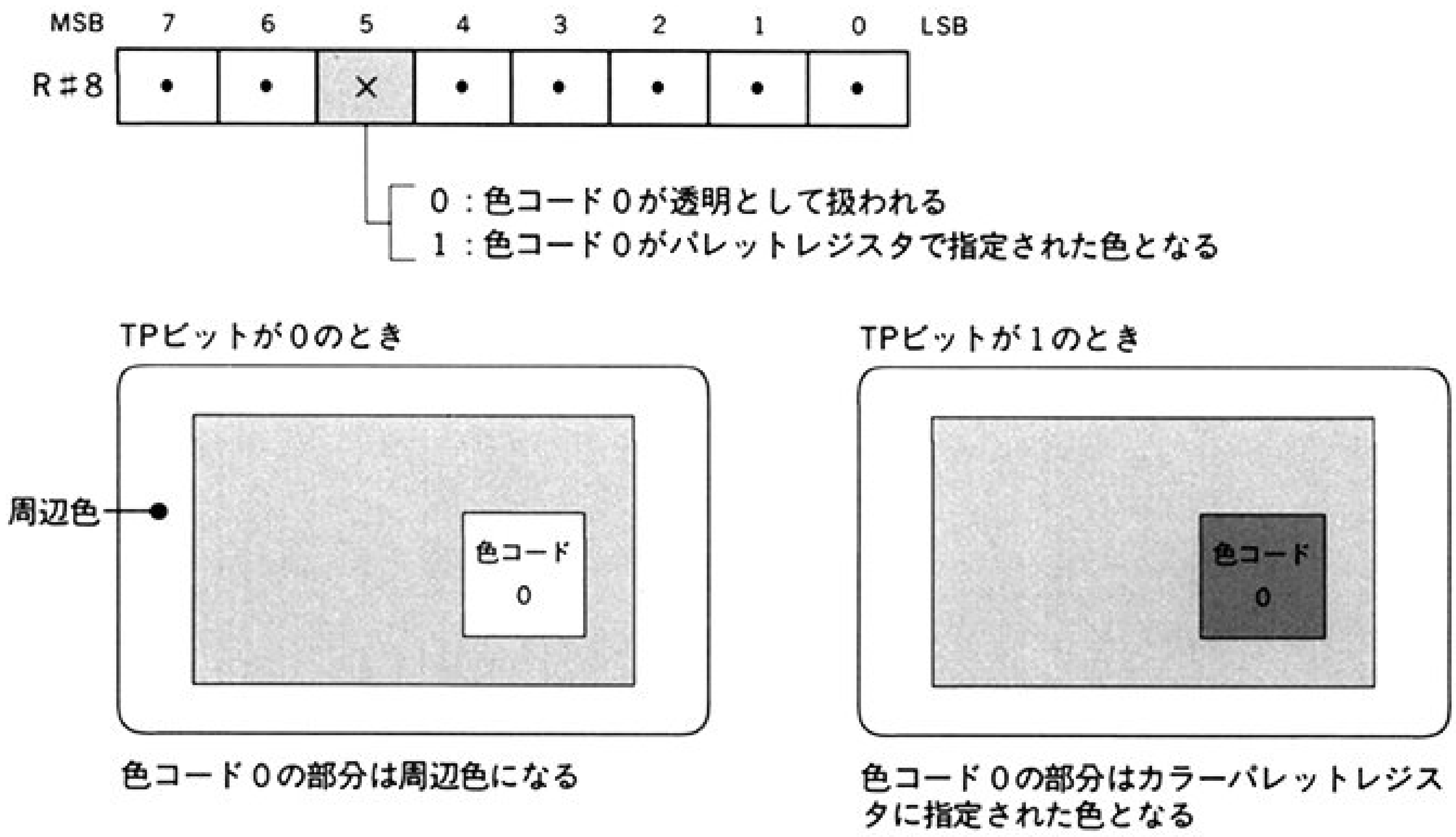


図4.55　色コード0の機能

●走査線位置による割り込みの発生

MSX-VIDEOでは，CRTが特定の走査線の表示を終えた瞬間に割り込みを発生させることができます．その場合は割り込みを発生させたい走査線番号をR#19に設定し，R#0のビット4を“1”にしておいてください(図4.56)．

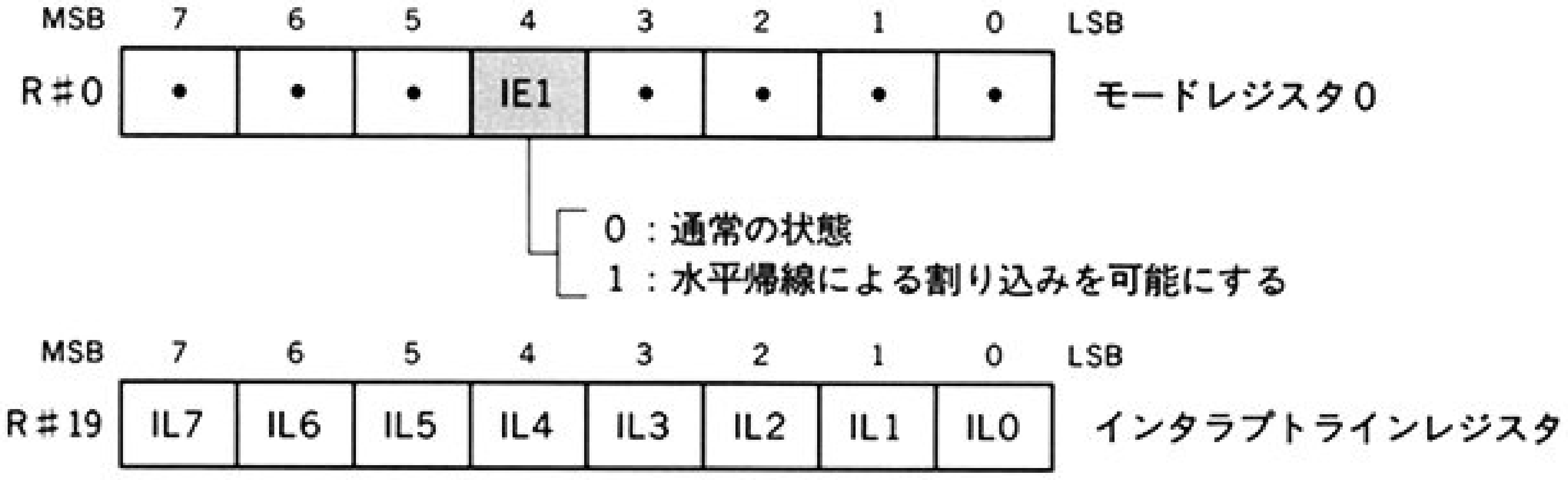


図4.56　走査線割り込みの発生

5章 スプライト

スプライトとは，8×8または16×16ドットの大きさを持つキャラクタパターンを画面上の任意の座標に表示する機能です．この機能を用いることにより，たとえばゲームのキャラクタの表示を簡単に行うことができます．

表示する時に指定するパラメータは，X座標，Y座標，キャラクタ番号，カラーコードの4つで，このデータをあらかじめ設定したスプライトアトリビュート・テーブルに書き込むことによりスプライトは表示されます．

MSX 2のスプライトには2種類のモードがあります．1つは従来のTMS9918互換モードであり，これをスプライトモード1といいます．もう1つは新しく追加された高機能なスプライトモードで，こちらをスプライトモード2といいます．以下，スプライト機能の概要をまとめた後，この両者を順に説明していきます．

5.1 スプライトの機能

1つの画面には，最高で32個のスプライトを表示することができます．

スプライトの大きさは8×8ドット，または16×16ドットの2種類あり，2つを同時に指定することはできません．また，一般にスプライトの1ドットの大きさは，グラフィックの1ドットと同じですが，GRAPHIC 5および6モード(両者とも512×212ドットの分解能)の場合には，横方向の大きさがグラフィックの2ドット分になります(結局，どのモードでも表示されるスプライトの絶対的な大きさは変わらない)．

その他のスプライトの機能に関しては，スプライトモードが1と2のどちらであるかによって変わってきます．スプライトのモードは，設定されている画面モードによって次のように自動的に選択されます．

スプライトモード1 が選択される	……	GRAPHIC 1	(SCREEN 1)
		GRAPHIC 2	(SCREEN 2)
		MULTI COLOR	(SCREEN 3)

スプライトモード２　……	GRAPHIC 3	(SCREEN 4)
が選択される	GRAPHIC 4	(SCREEN 5)
	GRAPHIC 5	(SCREEN 6)
	GRAPHIC 6	(SCREEN 7)
	GRAPHIC 7	(SCREEN 8)

5.2　スプライトモード１

スプライトモード１はMSX１のスプライトと同機能です．したがって，このモードを使っている限りはMSX１でも動作可能です．

5.2.1　最大表示数

32個のスプライトには，それぞれ０～31の番号が付けられていて，番号の若い方から高い優先順位が与えられています．画面の同一水平ライン上にスプライトが並んだ場合，優先度の高い順に４つまで表示され，５番目以降のスプライトの重なった部分は表示されません．

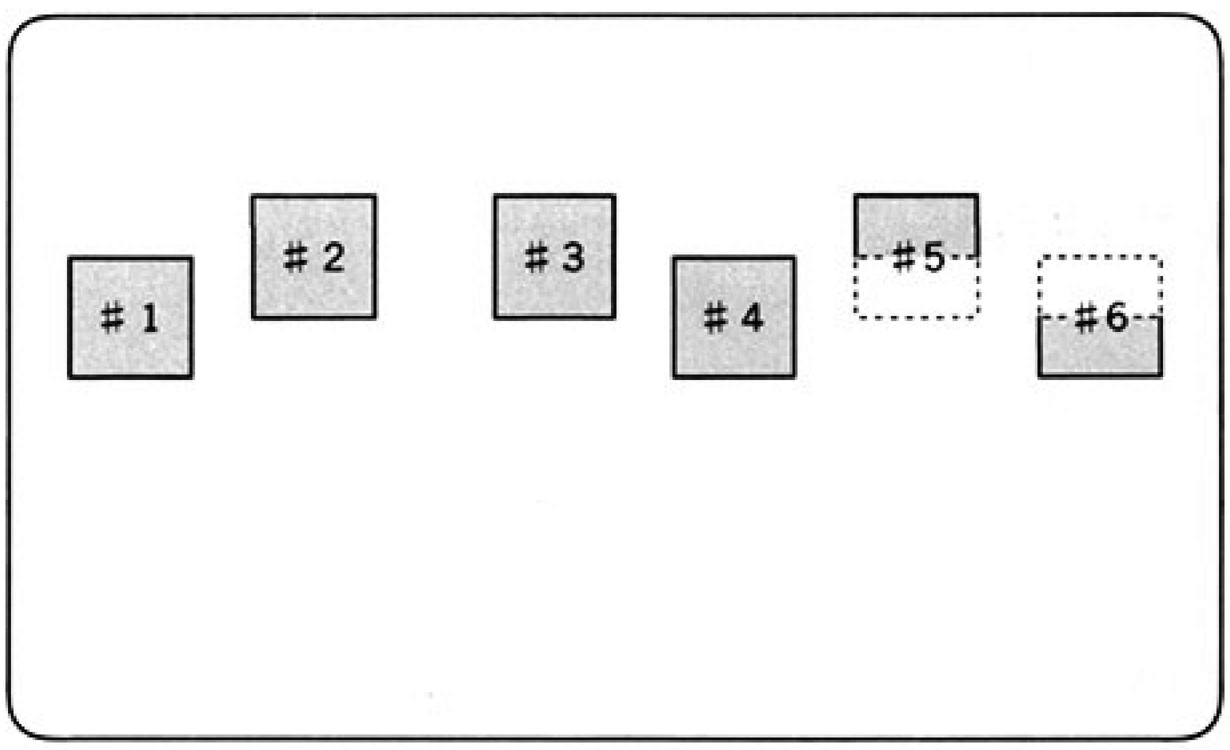


図4.57　スプライトの最大表示数(スプライトモード１)

5.2.2　スプライト表示のための諸設定

スプライトを表示するために必要な諸設定を以下に示します．

●スプライトのサイズの設定

８×８ドットまたは16×16ドットのどちらかを設定します(図4.58)．デフォルト(省略値)は８×８ドットになっています．

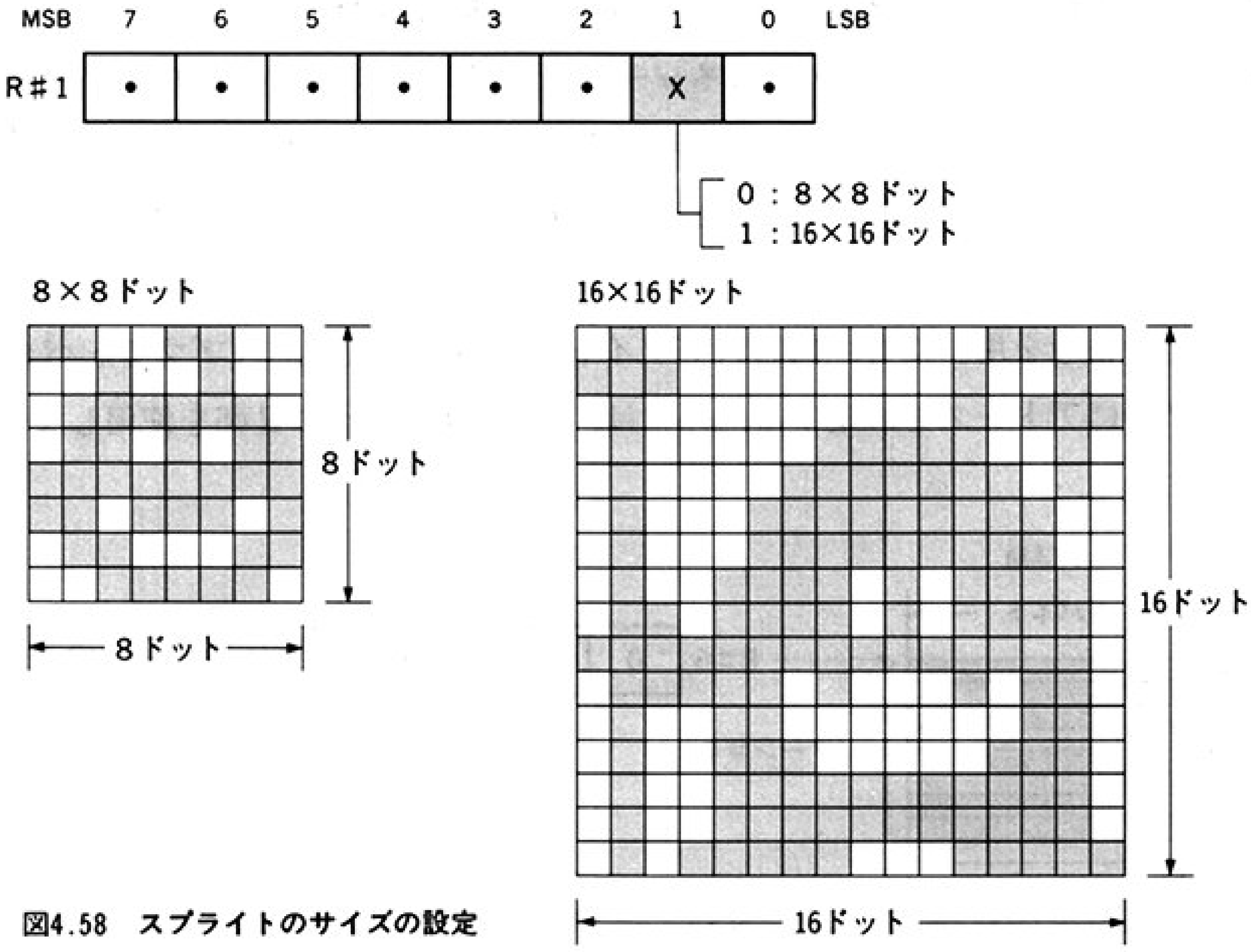


図4.58　スプライトのサイズの設定

●スプライトの拡大

スプライトの1ドットを画面の1ドットに対応させるか，または縦横2倍に拡大するかを設定します(図4.59)．デフォルトは1対1の対応になっています．

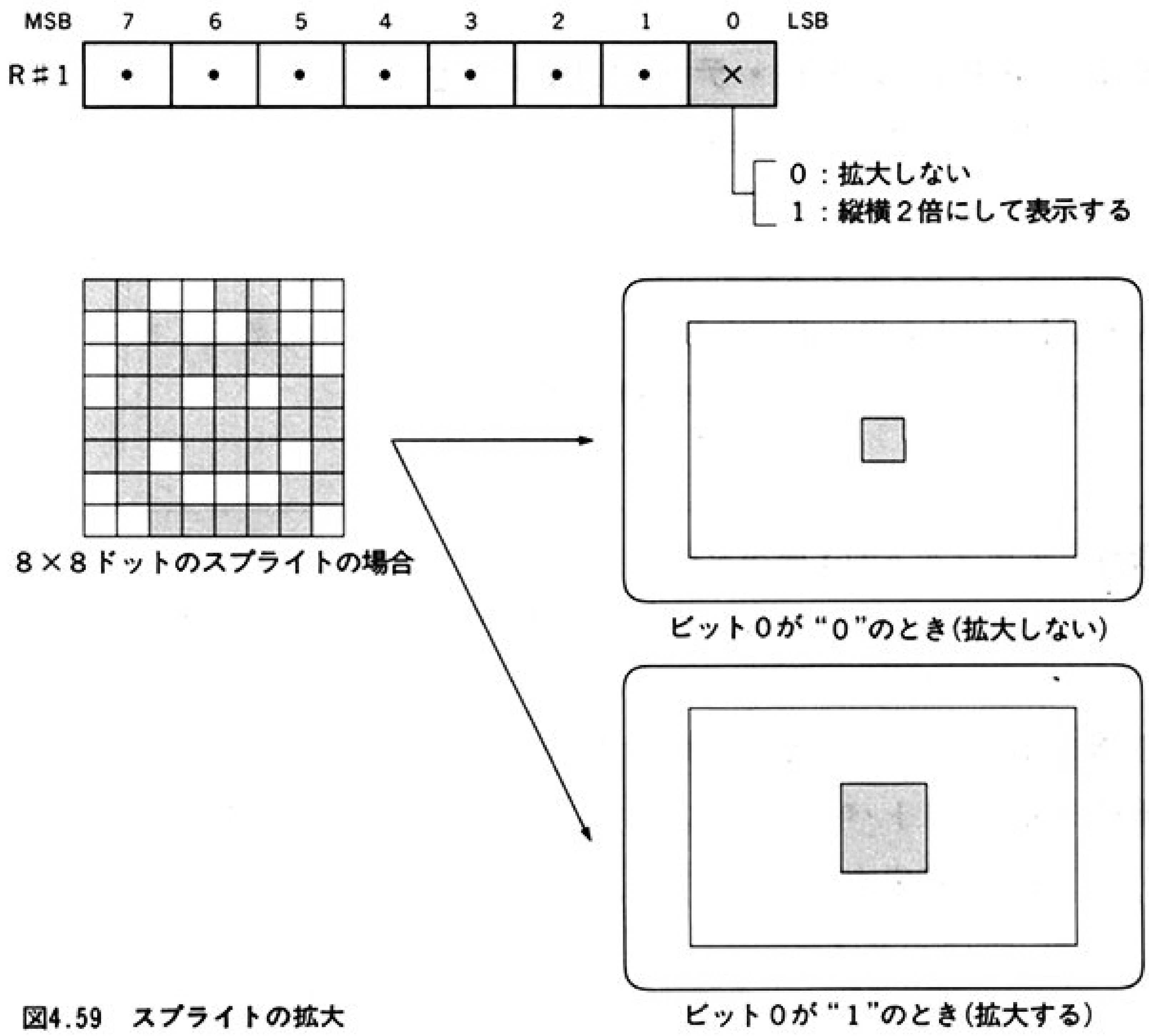


図4.59　スプライトの拡大

●スプライトパターン・ジェネレータテーブルの設定

VRAM 上のスプライトパターン・ジェネレータテーブルにスプライトのパターンを定義します．スプライトパターンは，8×8ドットの時256個，16×16の時64個まで定義可能です．各パターンには 0～255のパターン番号が与えられ，**図 4.60** のように VRAM に配置されます．16×16ドットの場合は，テーブルの先頭から 8×8サイズのパターンが 4 つずつ使われます．この時 4 つのパターンのどの番号を用いても，同じスプライトが指定されます．スプライトパターン・ジェネレータテーブルにアドレスを設定するためには，**図 4.60** のように R#6を使用します．

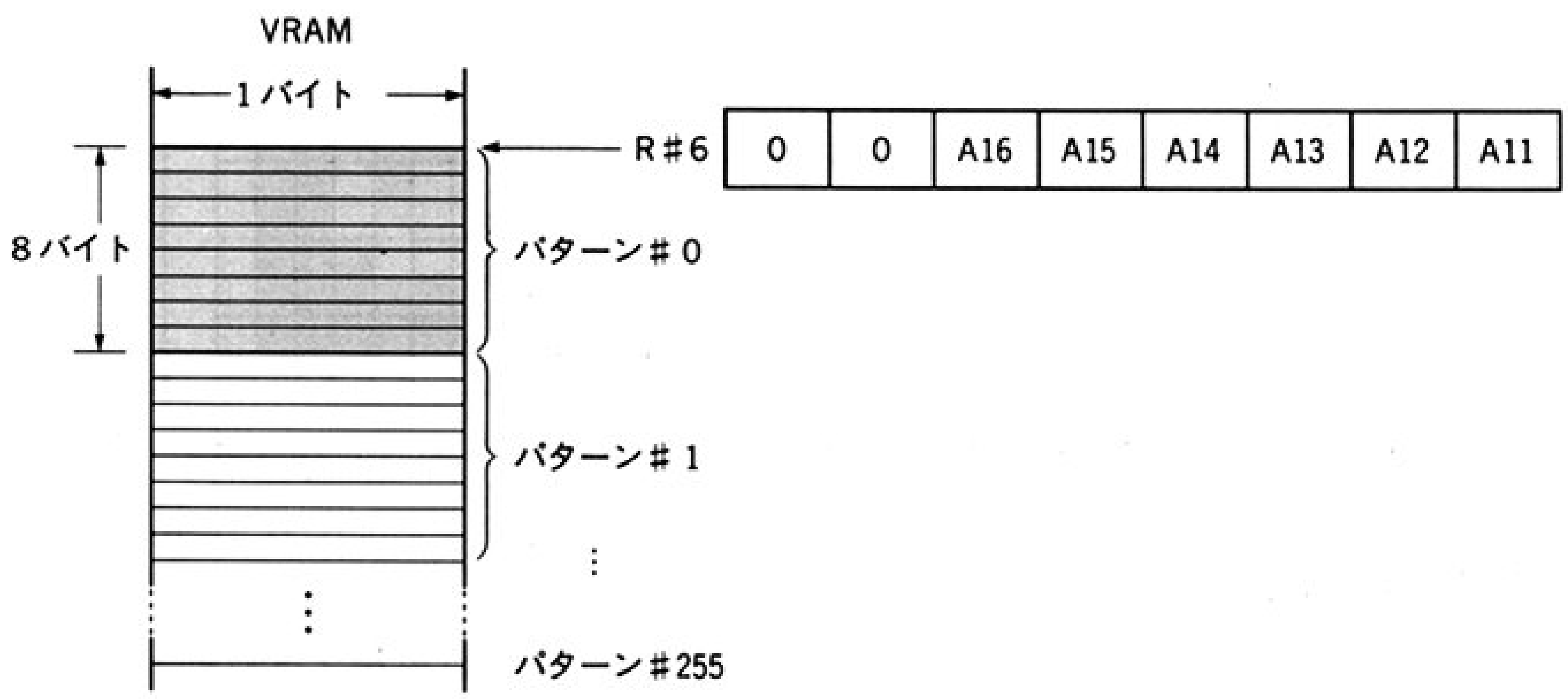


図4.60　スプライトパターン・ジェネレータテーブルの構造(スプライトモード1)

●スプライトアトリビュート・テーブルの設定

スプライトは，32枚の“スプライト面”にそれぞれ1つずつ表示され，そのスプライトの状態はスプライト面ごとに4バイトのメモリを用いて記録されます．スプライト面ごとの情報を記憶したこの領域を，スプライトアトリビュート・テーブルといいます．スプライトアトリビュート・テーブルは VRAM 上にとられ，R#5とR#11でその先頭アドレスを設定します．

アトリビュート中の4バイトは，以下の情報を表しています．

Y座標：スプライトのY座標を指定する．ただし，画面最上段のラインは0ではなく，255．この値を208(D0H)に設定すると，そのスプライト面以降のスプライトは表示されない

X座標：スプライトのX座標を指定する

パターン番号：スプライトパターン・ジェネレータテーブル中のどのキャラクタを表示するかを指定する

色コード：スプライトパターンのビットが“1”である部分の色(パレット番号)を指定する

EC：このビットを“1”にすると，スプライトは左へ32ドット分シフトされる．この機能を使用すると，画面の左端からスプライトを1ドットずつ出現させることができる

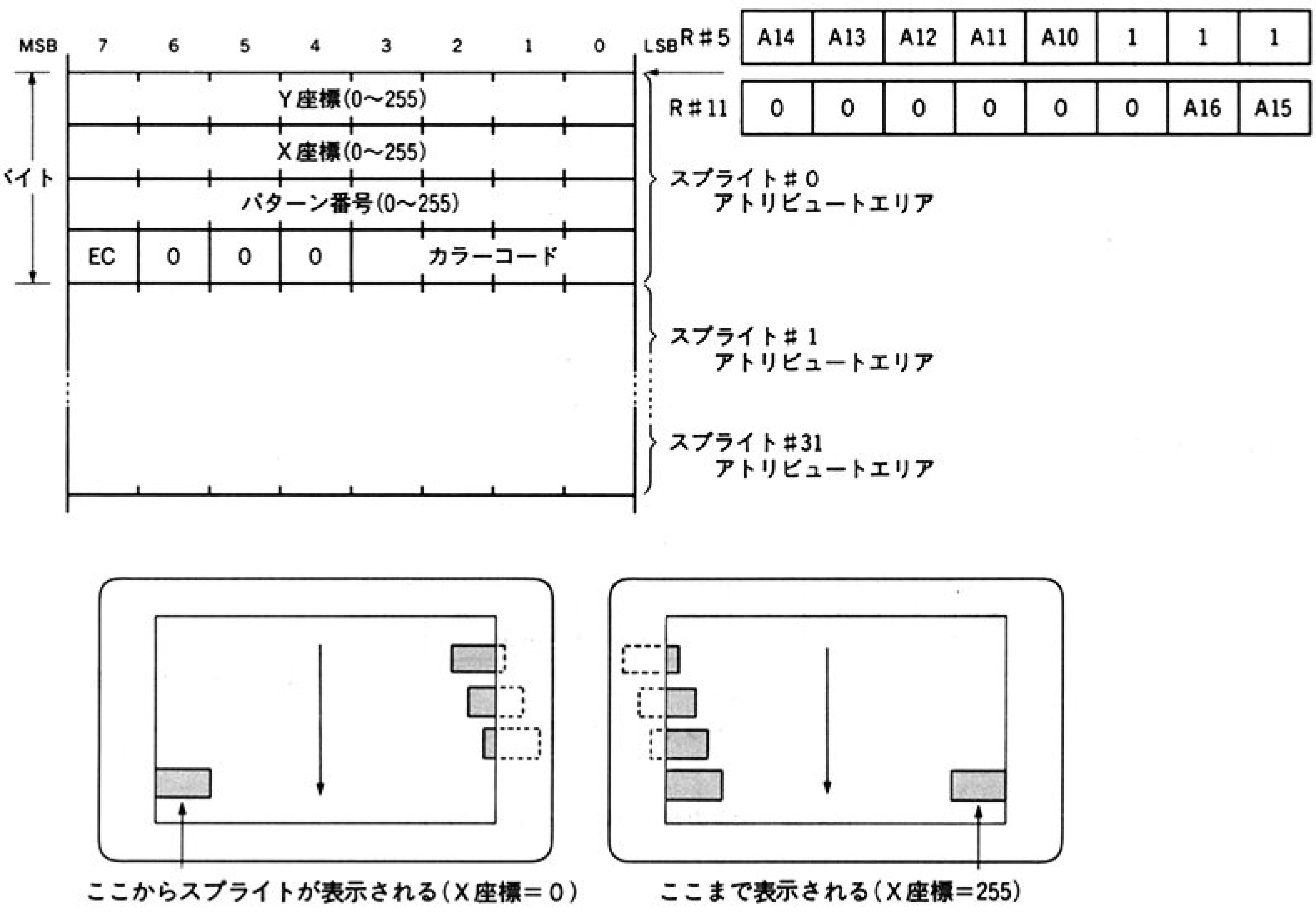


図4.61 スプライトアトリビュート・テーブルの構造(スプライトモード1)

5.2.3 スプライトの衝突判定

2個のスプライトが衝突した時，S#0のビット5が“1”になって衝突の発生を知らせます．ここで“衝突”とは，“透明色”ではない色のスプライトパターン中の“1”のビット同士が同一座標を占めた場合を指します(図4.62)．

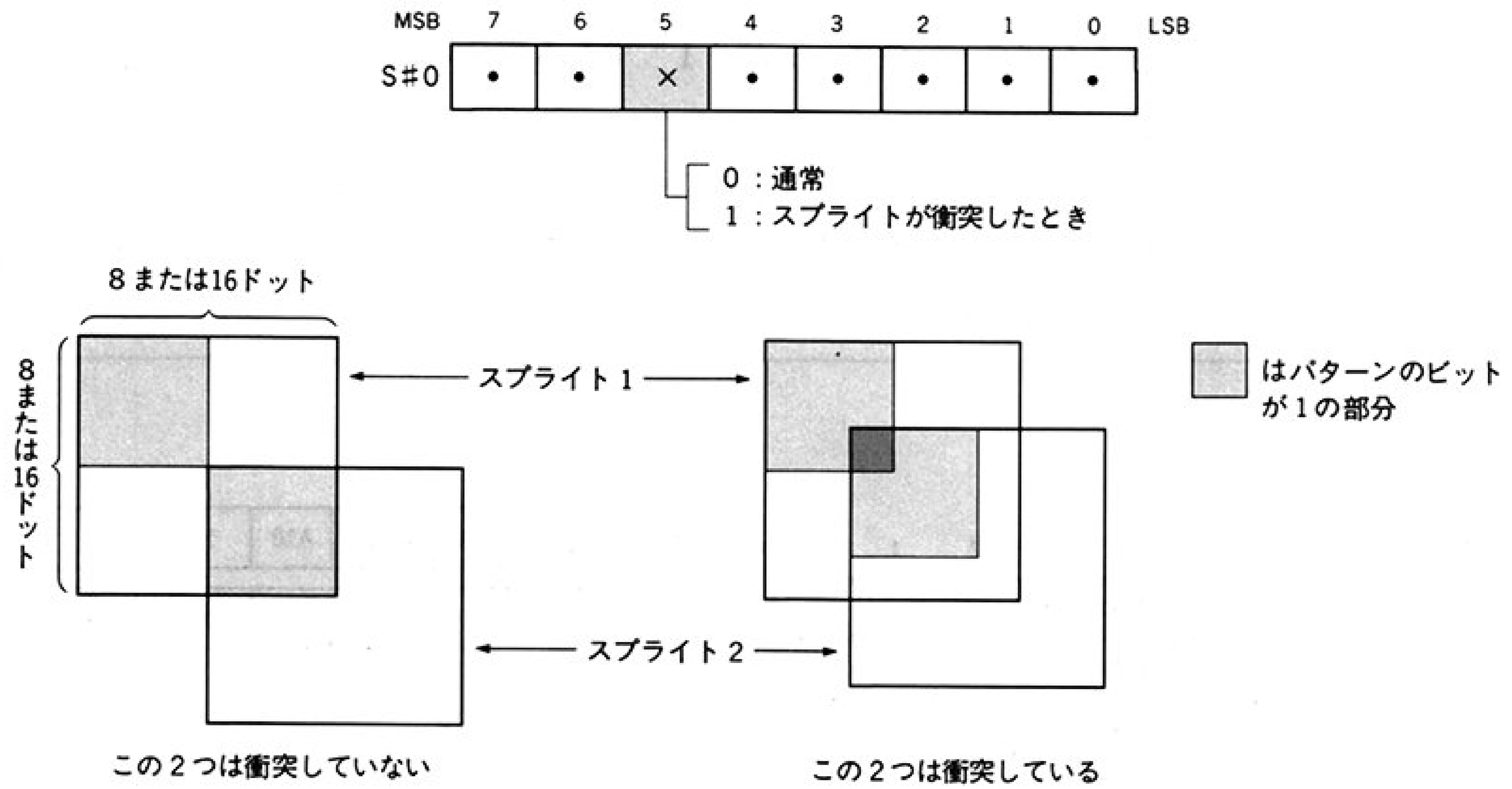


図4.62　スプライトの衝突(スプライトモード1)

同一ラインに5個以上のスプライトが並んだ時には，S#0のビット6が“1”となり，さらにS#0の下位5ビットに，5番目(表示できなくなった部分)のスプライトの番号がセットされます．

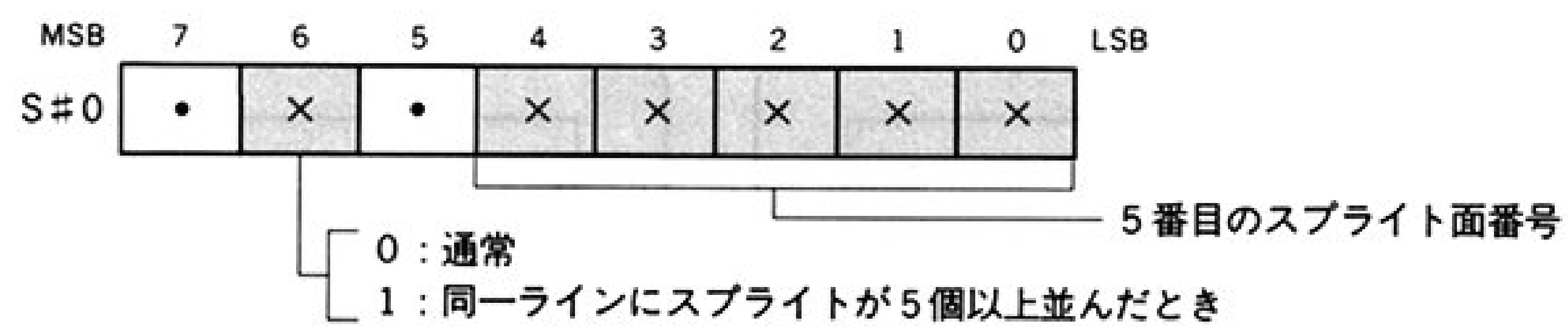


図4.63　衝突の判定(スプライトモード1)

5.3　スプライトモード2

スプライトモード2は，MSX-VIDEOで新設されたモードです．したがって，TMS 9918との互換性はありません．

5.3.1　最大表示数

1画面に表示できるスプライトの数はやはり32個ですが，画面の同一水平ライン上には優先度の高い順に8つまで表示できます．9番目以降のスプライトの重なった部分は表示されません．

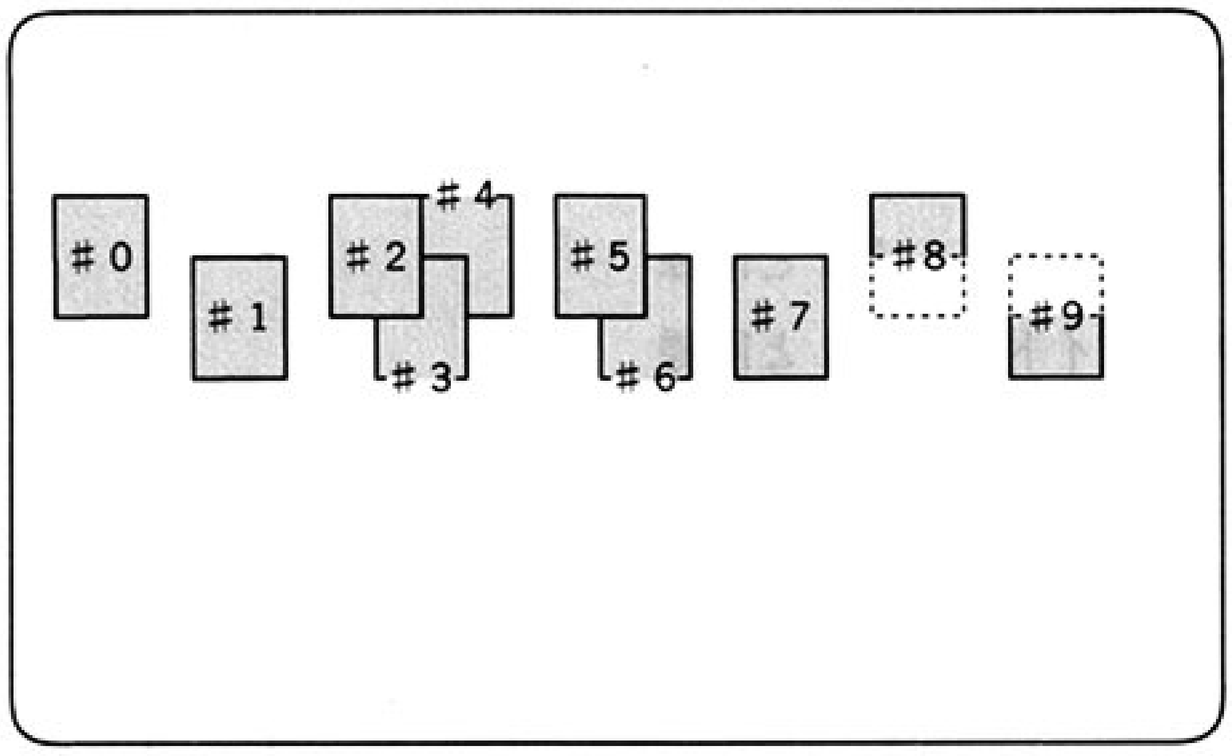


図4.64　スプライトの最大表示数(スプライトモード2)

5.3.2　スプライト表示のための諸設定

●スプライトサイズ………スプライトモード1と同様.

●スプライトの拡大………スプライトモード1と同様.

●スプライト表示のON/OFF

スプライトモード2では，R#8のビット1によってスプライトの表示をON／OFFすることができます.スプライトがまったく画面に現れない場合，このビットが“1”に設定されている可能性があります.

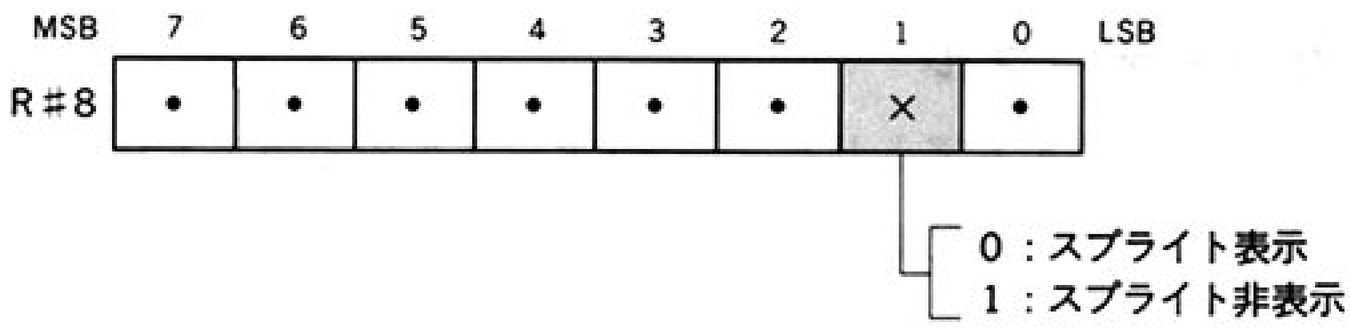


図4.65　スプライト表示の指定

●パターンジェネレータ・テーブルの設定　……　スプライトモード1と同様.

●スプライトアトリビュート・テーブル

スプライトモード2ではスプライトの横1ラインごとに異なった色を付けることが可能となり，そのため色情報はスプライトアトリビュート・テーブルから独立して，次に述べるスプライトカラーテーブルに記憶されることになりました．スプライトアトリビュート・テーブルには，以下の3種類の情報が格納されます(図4.66).

Y座標：この値を216(D8H)に設定すると，そのスプライト面以降のスプライトは表示されなくなる．それ以外はスプライトモード１と同様

X座標：スプライトモード１と同様

パターン番号：スプライトモード１と同様

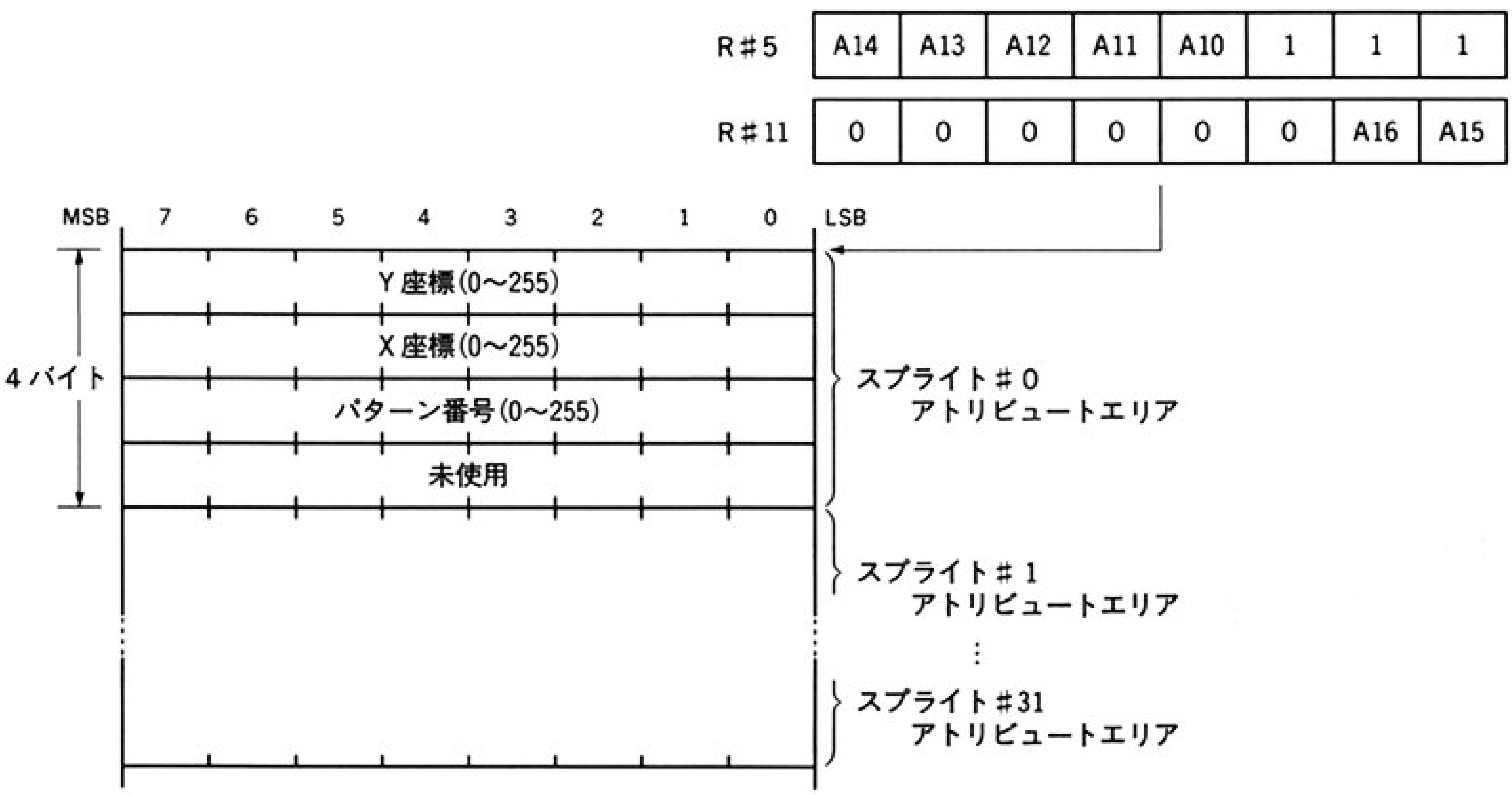


図4.66　スプライトアトリビュート・テーブルの構造(スプライトモード２)

●スプライトカラーテーブル

カラーテーブルは，スプライトアトリビュート・テーブルの先頭アドレス−512の位置に自動的に設定されます．各スプライト面に対して16バイトの領域がとられ，それぞれスプライトの１ラインごとに以下のような設定を行います．

色コード：１ラインごとに色指定可能．

EC：スプライトモード１のアトリビュートテーブルのECビットと同様，“１”であるときはスプライトの表示位置が左へ32ドット分シフトされる．これも１ラインごとに指定可能．

CC：CCビットが“１”の場合，「このスプライトよりも優先順位が高く，かつCCビットが“０”で，最もこのスプライト面に近い」スプライトと等しい優先順位が得られる．等しい優先順位を持つスプライトが重なった場合には，その両者の色コードのOR(論理和)をとったものが表示される．この場合，重なっても衝突は発

生しない(図 4.68 参照).

IC：このビットが“1”であるスプライト(の1ライン)は，他のスプライトとの衝突を発生しない.

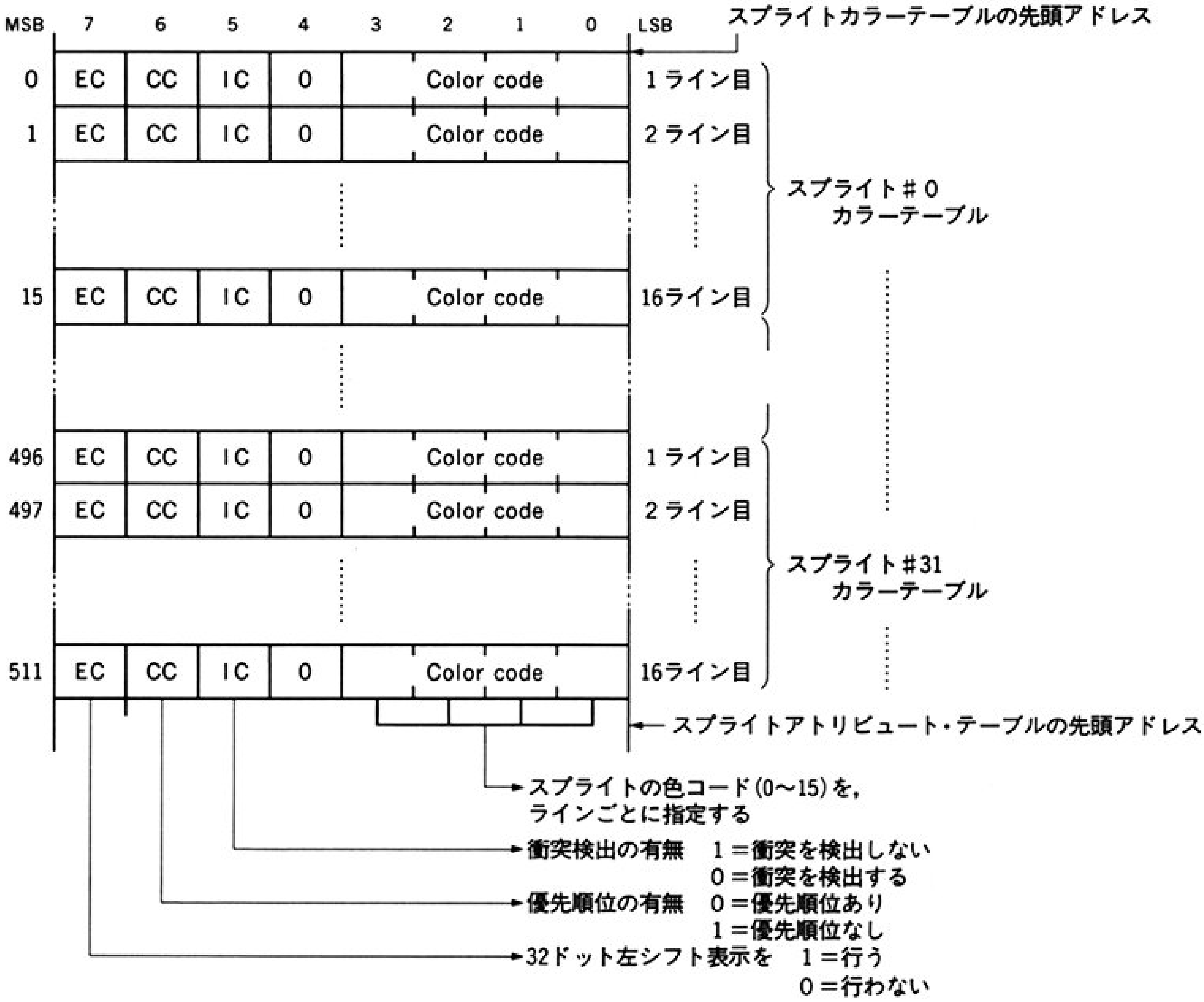
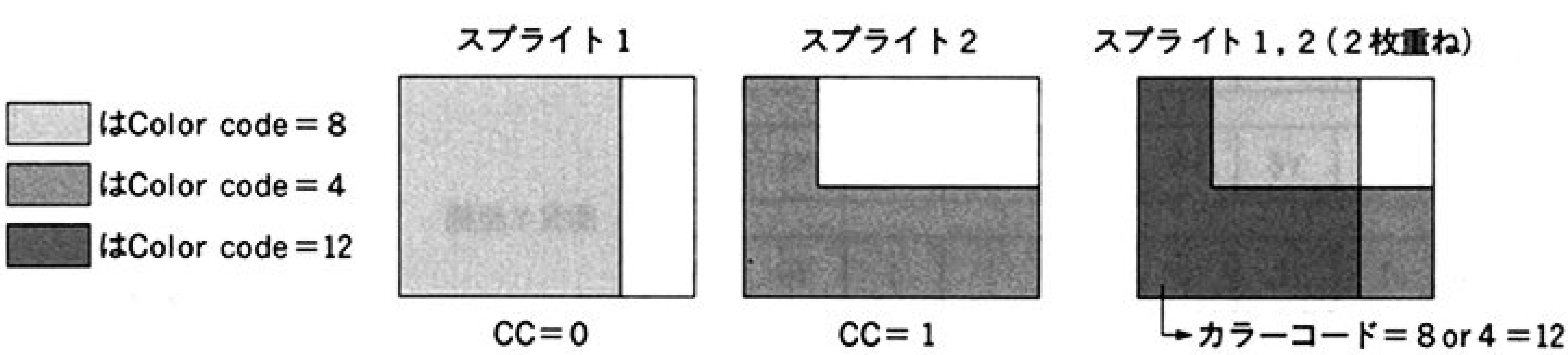


図4.67　スプライトカラーテーブルの構造(スプライトモード2)



注1) CCを1にしたスプライトは，そのスプライトより若い番号で最もそのスプライトに近いスプライトのCC＝0の部分とパターンが重なっても衝突の検出は，行われない．
2) CC＝1のスプライトを表示させるには，表示させたいスプライトより若い番号のスプライトのccbitを0にしなければならない．

図4.68　cc bitの意味

5.3.3 スプライトの衝突判定

スプライトモード2における"衝突"とは，スプライトの表示色が透明でなく，かつCCビットが0であるライン上の"1"のビット同士が重なった場合をいいます．2つのスプライトが衝突した場合はS#0のビット5が"1"になり，衝突が検出できます(図4.69)．この時スプライトモード1と異なり，衝突の発生した座標を図4.70のようにS#3～S#6によって知ることができます．ただし，これらのレジスタにより得られる座標は，実際に衝突が起こった座標ではありません．それを求めるには式4.5で算出してください．なお，S#3～S#6はS#5を読み出した時にリセットされます．

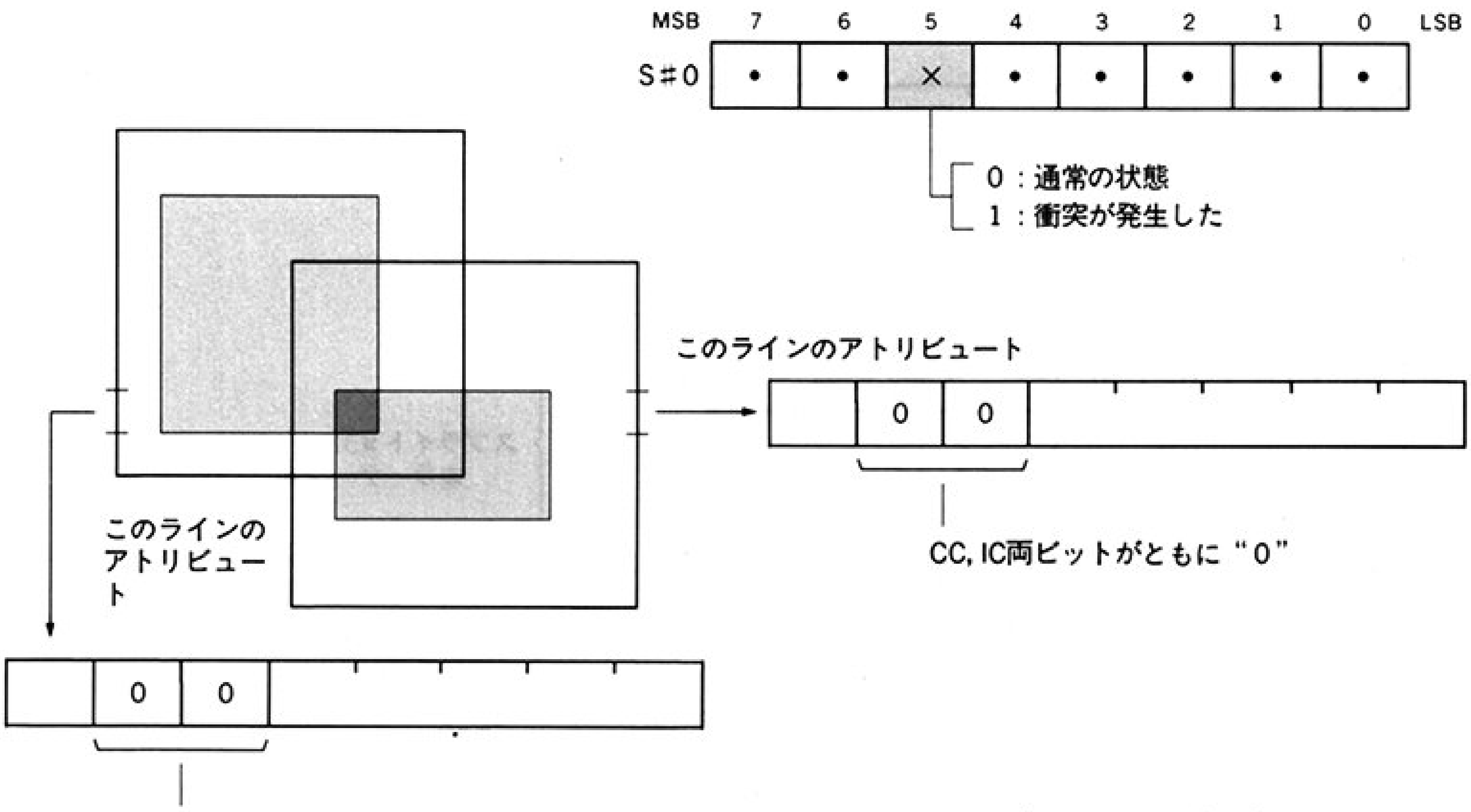


図4.69　スプライトの衝突(スプライトモード2)

	MSB 7	6	5	4	3	2	1	0 LSB	
S#3	X7	X6	X5	X4	X3	X2	X1	X0	衝突X座標
S#4	1	1	1	1	1	1	1	X8	
S#5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	衝突Y座標
S#6	1	1	1	1	1	1	Y9	Y8	

図4.70　衝突座標の読み出し

(衝突が起こったX座標)＝(S#3，S#4のX座標)－12
(衝突が起こったY座標)＝(S#5，S#6のY座標)－8

式4.5　実衝突座標の算出

同一水平ライン上に9個以上のスプライトが並んだ時には，S#0のビット6が“1”になり，さらにS#0の下位5ビットに9番目に優先順位の低いスプライト面の番号が入ります(図4.71).

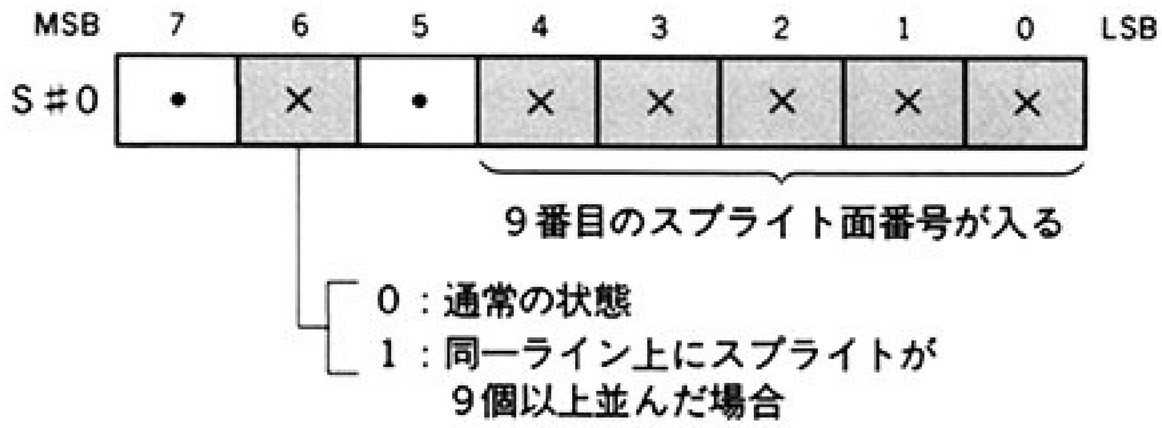


図4.71　スプライトの衝突(スプライトモード2)

6章 VDPコマンドの使用法

MSX-VIDEOではGRAPHIC 4～GRAPHIC 7モードにおいて，基本的なグラフィック処理をハードウェアによって行えるようになっており，それらをまとめてVDPコマンドと呼んでいます．VDPコマンドによるグラフィックの描画は，必要なパラメータを設定した後にコマンドを起動するのみでよく，ソフトウェアの負担軽減およびグラフィックのスピードアップの点で大きな効果が得られます．本章ではこのVDPコマンドについて説明します．

6.1 VDPコマンドの座標系

VDPコマンドを実行する際，ソース(転送元)あるいはデスティネーション(転送先)の位置はすべて図4.72のような(X,Y)座標で表されます．また同図からわかるように，コマンド実行時にはページの区別はいっさいなくなり，128Kバイトの VRAM 全体が1つの大きな座標系の中に置かれます．

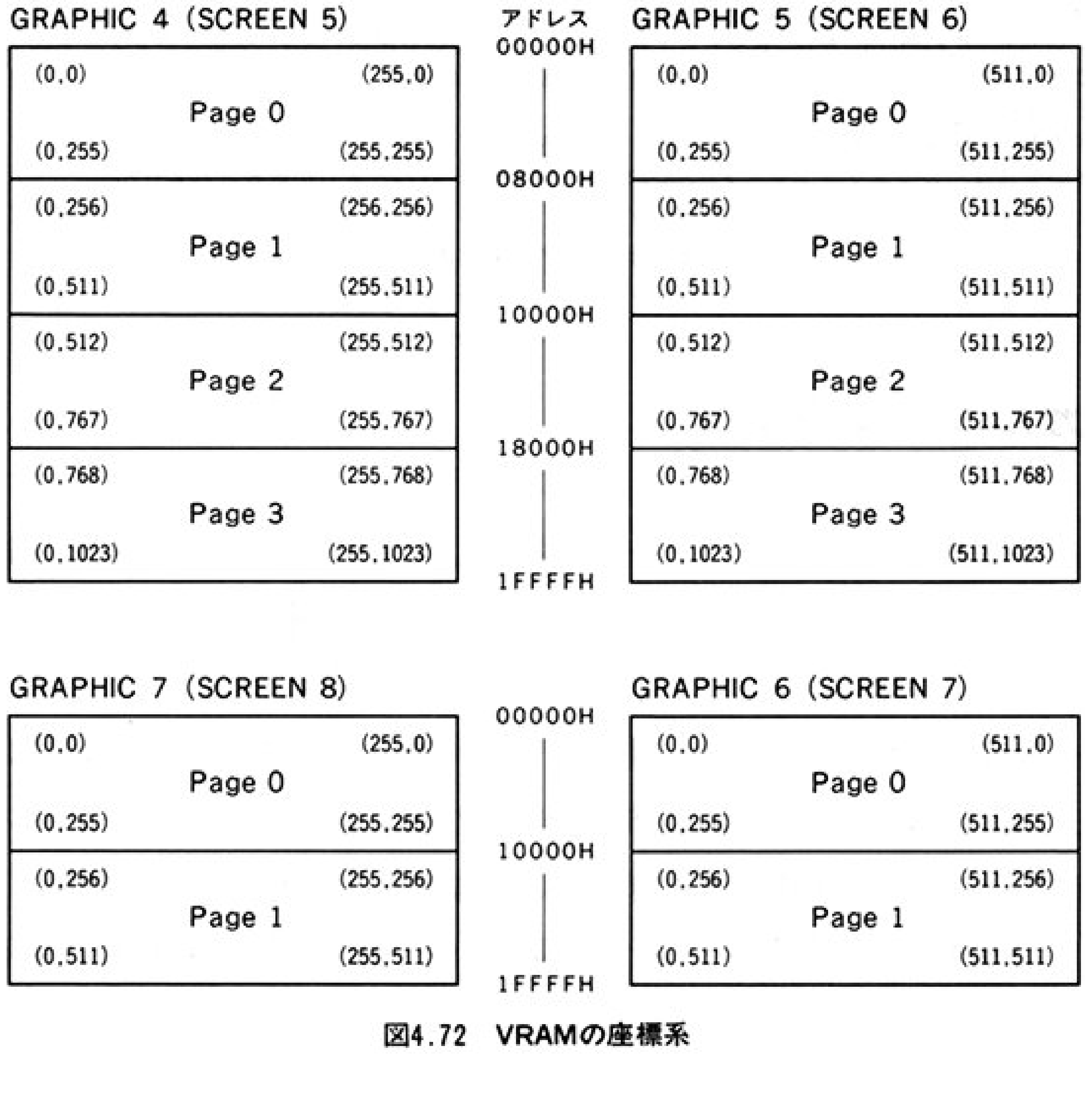


図4.72 VRAMの座標系

6.2 VDPコマンドの種類

MSX-VIDEOで実行できるコマンドには，**表4.5**に示す12種類があります．

コマンド名	転送先	転送元	転送単位	ニーモニック	R#46(上位4ビット)			
High speed move (高速移動)	VRAM	CPU	バイト	HMMC	1	1	1	1
	VRAM	VRAM	バイト	YMMM	1	1	1	0
	VRAM	VRAM	バイト	HMMM	1	1	0	1
	VRAM	VDP	バイト	HMMV	1	1	0	0
Logical move (論理移動)	VRAM	CPU	ドット	LMMC	1	0	1	1
	CPU	VRAM	ドット	LMCM	1	0	1	0
	VRAM	VRAM	ドット	LMMM	1	0	0	1
	VRAM	VDP	ドット	LMMV	1	0	0	0
Line(描線)	VRAM	VDP	ドット	LINE	0	1	1	1
Search(探索)	VRAM	VDP	ドット	SRCH	0	1	1	0
Pset(描点)	VRAM	VDP	ドット	PSET	0	1	0	1
Point	VDP	VRAM	ドット	POINT	0	1	0	0
未使用	—	—	—	—	0	0	1	1
	—	—	—	—	0	0	1	0
	—	—	—	—	0	0	0	1
Stop	—	—	—	—	0	0	0	0

・MSX-VIDEOは，レジスタR#46(Command register)にデータが書き込まれると，ステータスレジスタS#2のビット0(CE／Command Execute)を1にセットしてからコマンドの実行を開始します．必要なパラメータは，コマンド実行前にレジスタR#32～R#45にセットされていなければなりません．
・コマンドの実行が終了するとCEは0となります．
・コマンドの実行を途中で中断するには，STOPコマンドを実行して下さい．
・コマンドの動作は，ビットマップモード(GRAPHIC 4～GRAPHIC 7)のときしか保証されません．

表4.5　VDPコマンド一覧

6.3 ロジカルオペレーション

コマンドを実行する際，VRAM上のデータと指定したデータの間で，種々の論理演算(ロジカルオペレーション)を行うことができます．実際には，各演算は**表4.6**に示したような法則にしたがって行われます．

表中SCはソースの色，DCはデスティネーションの色を表す数値です．IMP，AND，OR，EOR，NOTは，SCとDCにそれぞれの演算を施した結果をデスティネーションに書き込みます．また，頭に“T”のついた演算では，SC=0に相当するドットは演算の対象とならず，DCのまま残ります．この演算を使用すると，2つの図形の色の部分だけを重ね合わせることが可能に

なり，アニメーションなどに威力を発揮します．

この演算の例を**リスト 4.7** に示します．

演算名	Operation（演算動作）	L03	L02	L01	L00
IMP	DC=SC	0	0	0	0
AND	DC=SC×DC	0	0	0	1
OR	DC=SC+DC	0	0	1	0
EOR	DC=$\overline{SC}$×DC+SC×$\overline{DC}$	0	0	1	1
NOT	DC=$\overline{SC}$	0	1	0	0
—		0	1	0	1
—		0	1	1	0
—		0	1	1	1
TIMP	if SC=0 then DC=DC else DC=SC	1	0	0	0
TAND	if SC=0 then DC=DC else DC=SC×DC	1	0	0	1
TOR	if SC=0 then DC=DC else DC=SC+DC	1	0	1	0
TEOR	if SC=0 then DC=DC else DC=$\overline{SC}$×DC+SC×$\overline{DC}$	1	0	1	1
TNOT	if SC=0 then DC=DC else DC=$\overline{SC}$	1	1	0	0
—		1	1	0	1
—		1	1	1	0
—		1	1	1	1

※SC=Source Color code
※DC=Destination Color code
※EOR=Exclusive OR

表4.6　ロジカルオペレーション一覧

リスト4.7　Tつき演算の例

```
1000 '================================================
1010 '  List 4.7   logical operation with T
1020 '================================================
1030 '
1040 SCREEN8 : COLOR 15,0,0 : CLS
1050 DIM A%(3587)
1060 '
1070 LINE (50,50)-(60,100),48,B : PAINT (51,51),156,48
1080 CIRCLE (55,30),30,255 : PAINT(55,30),240,255
1090 COPY(20,0)-(90,100) TO A%
1100 CLS
1110 '
1120 R=RND(-TIME)
1130 FOR Y=0 TO 100 STEP 3
1140   X=INT(RND(1)*186)
1150   COPY A% TO (X,Y),,TPSET
1160 NEXT
1170 '
1180 GOTO 1180
```

6.4　領域指定

AREA-MOVEに属するコマンドは，任意の長方形に囲まれた領域内の画像データを転送するものです．転送元領域は，**図4.73**のように長方形の1つの頂点と各辺の長さによって指定され，SX，SYは転送元長方形の基準点，NX，NYは各辺の長さ(ドット数)，DIX，DIYの2ビットはデータ転送の方向を表します(DIXとDIYの意味はコマンドの種類によって異なる)．DX，DYには転送先領域の基準点を指定します．

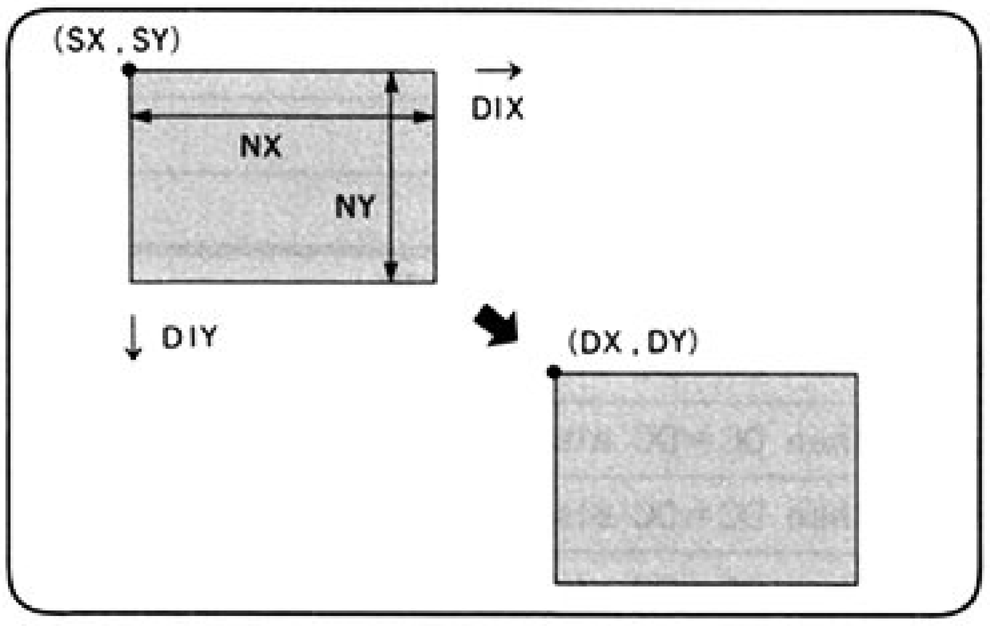


図4.73　領域の指定

6.5　各コマンドの使用法

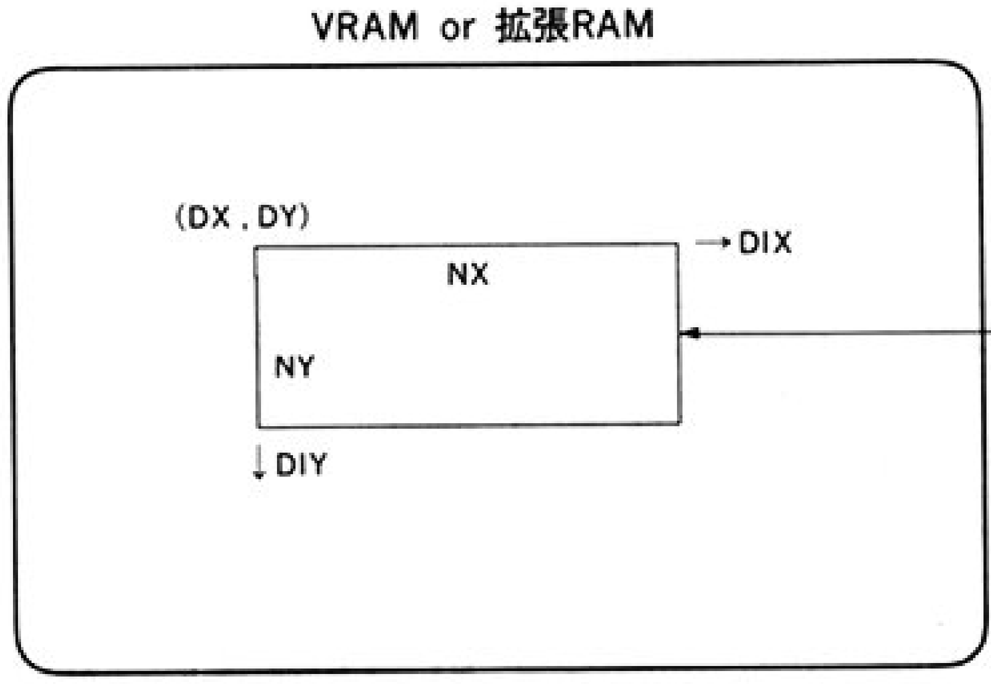
コマンドは大きく分けて高速転送コマンド，論理転送コマンド，描画コマンドの3種類あり，使用法もさまざまです．次にそれらを1つ1つ解説します．

6.5.1　HMMC(CPU → VRAM 高速転送)

VRAMの指定された領域にCPUからデータをバイト単位で転送します(**図4.74**)．ロジカルオペレーションは指定できません．このHMMCをはじめとする高速転送コマンドでは，データはバイト単位で転送されます．したがって，GRAPHIC 4～6のモードではそれぞれX座標の下位1ビット，2ビット，1ビットが参照されない点に注意してください(**図4.75**)．

図4.76に示すパラメータをレジスタに設定します．この時点で，CPU側から転送するデータの最初の1バイトだけはR#44に書き込んでおきます．R#46にコマンドコードF0Hを書き込むとコマンドの実行が開始され，MSX-VIDEOはR#44のデータを受け取ってVRAMに書き込み，その後CPU側からのデータ待ち状態になります．

CPU側では，第2バイト以降のデータをR#44に書き込んでいきます．ただしS#2のTRビットを参照し，MSX-VIDEOがデータ受け入れ可能な状態(TRビットが"1")になるのを待ってデ



MXD：転送先　メモリ選択　　0＝VRAM，1＝拡張RAM

NX　：X方向転送ドット数　(0～511) ※
NY　：Y方向転送ドット数　(0～1023)

DIX　：基準点からのNXの方向　　0＝右，1＝左
DIY　：基準点からのNYの方向　　0＝下，1＝上

DX　：転送先基準点　X座標　(0～511) ※
DY　：転送先基準点　Y座標　(0～1023)

CLR(R#44：Color register)：転送データの第1バイト

※DX，NXともに，GRAPHIC 4/GRAPHIC 6モードのときは下位1ビット，
GRAPHIC 5モードのときは下位2ビットが無視される．

図4.74　HMMCコマンドの動作

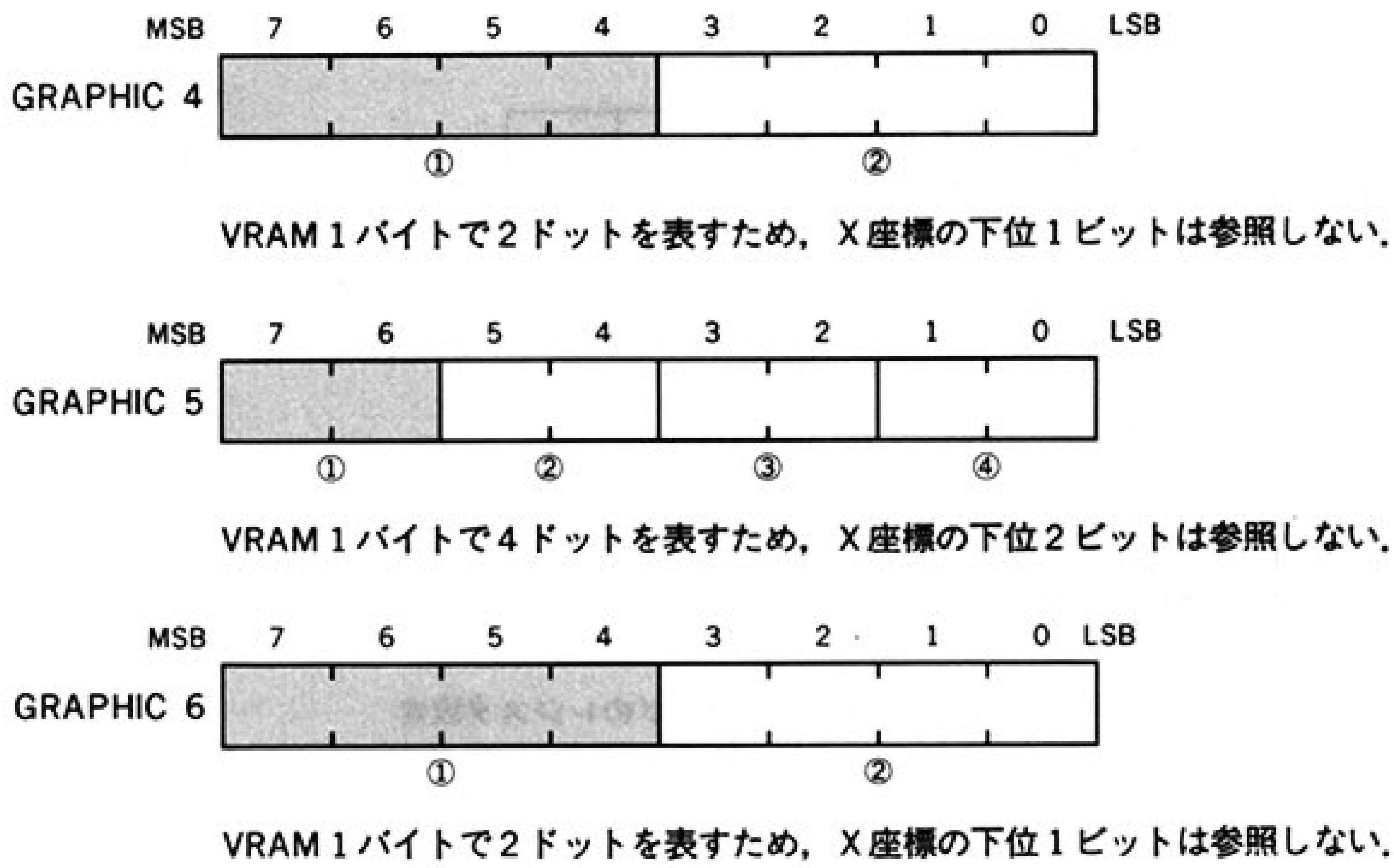


図4.75　参照されないドット

ータを送らなくてはいけません．S#2のCEビットが“0”になるとすべてのデータ転送が終わったことを示します(**図4.77**)．**リスト4.8**にHMMCのサンプルを示します．

▶HMMCレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX } 転送先基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY } 転送先基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向転送ドット数
R#41	0	0	0	0	0	0	0	NX8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数
R#43	0	0	0	0	0	0	NY9	NY8	
R#44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	CLR (GRAPHIC4, 6)
	X=2N				X=2N+1				(N=0, 1, ……, 127)
	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	CLR (GRAPHIC5)
	X=4N		X=4N+1		X=4N+2		X=4N+3		(N=0, 1, ……, 127)
	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	CLR (GRAPHIC7)
	1バイト/ドット								

転送データ（CLR: GRAPHIC4, 6 / GRAPHIC5 / GRAPHIC7）

	7	6	5	4	3	2	1	0	
R#45	0	−	MXD	−	DIY	DIX	−	−	ARG (Argument register)

- DIX → 転送方向（X）
- DIY → 転送方向（Y）
- MXD → 転送先メモリ選択

▶HMMCコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	1	1	1	1	−	−	−	−	CMR

図4.76　HMMCコマンドのレジスタ設定

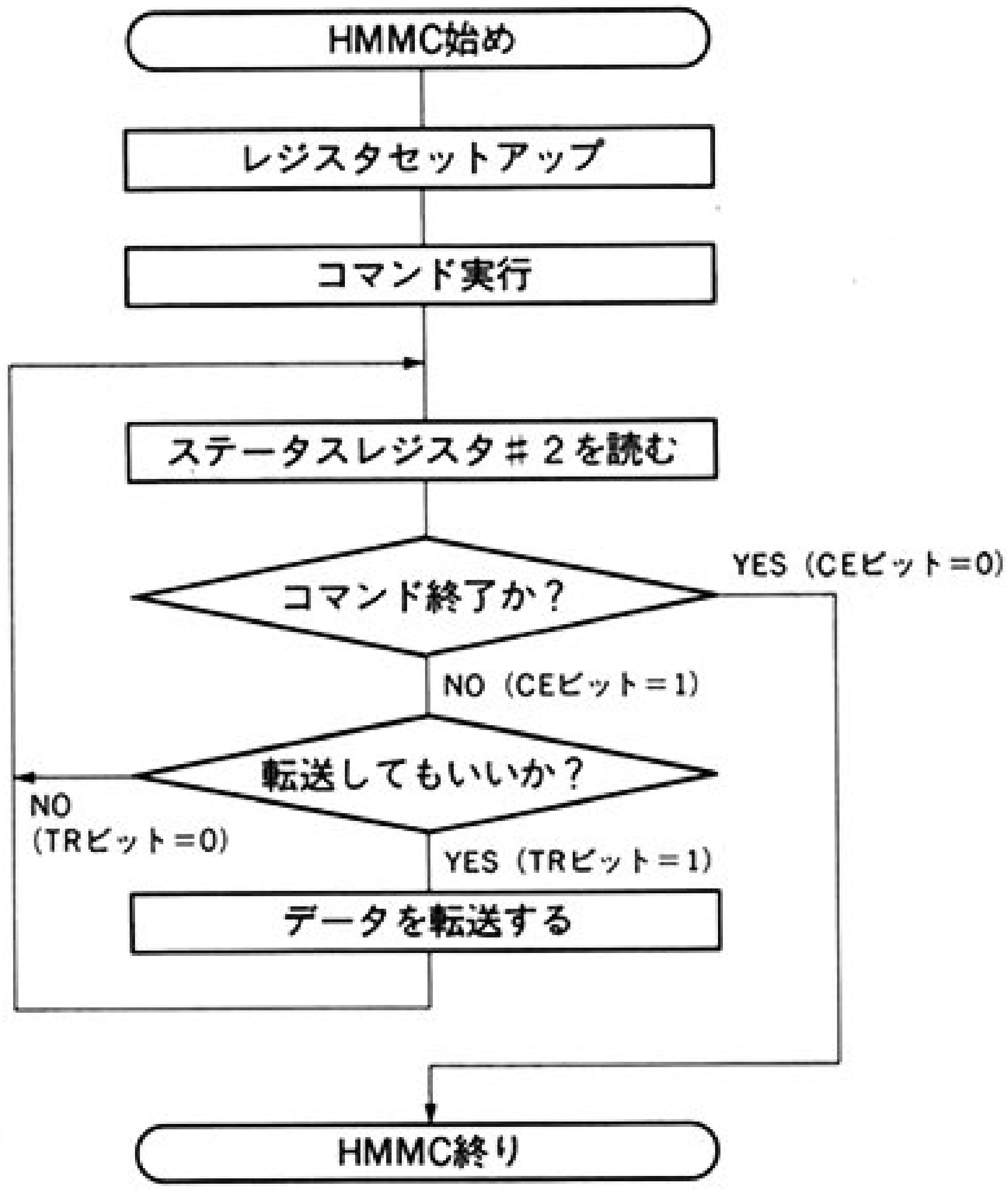


図4.77 HMMCコマンド実行のフローチャート

リスト4.8 HMMCコマンド実行例

```
;============================================================
;  List 4.8    HMMC sample
;                  to use, set H,L,D,E,IX and go
;                  RAM [IX] ---> VRAM (H,L)-(D,E)
;============================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

HMMC:   DI                              ;disable interrupt
        CALL    WAIT.VDP                ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C                       ;C := PORT#1's address
        LD      A,36
        OUT     (C),A
        LD      A,17+80H
        OUT     (C),A                   ;R#17 := 36

        INC     C
        INC     C                       ;C := PORT#3's address
        XOR     A
        OUT     (C),H                   ;DX
        OUT     (C),A
        OUT     (C),L                   ;DY
        OUT     (C),A
```

```
        LD      A,H             ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JR      NC,HMMC1
        LD      D,00000000B
        NEG
HMMC1:  LD      H,A             ;H := NX , D := DIX

        LD      A,L             ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JR      NC,HMMC2
        LD      E,00000000B
        NEG
HMMC2:  LD      L,A             ;L := NY , E := DIY

        XOR     A
        OUT     (C),H           ;NX
        OUT     (C),A
        OUT     (C),L           ;NY
        OUT     (C),A
        LD      H,(IX+0)
        OUT     (C),H           ;first DATA
        LD      A,D
        OR      E
        OUT     (C),A           ;DIX and DIY
        LD      A,0F0H
        OUT     (C),A           ;HMMC command

        LD      A,(RDVDP)
        LD      C,A             ;C := PORT#1's address
LOOP:   LD      A,2
        CALL    GET.STATUS
        BIT     0,A             ;check CE bit
        JR      Z,EXIT
        BIT     7,A             ;check TR bit
        JR      Z,LOOP
        INC     IX
        LD      A,(IX+0)
        OUT     (C),A
        JR      LOOP

EXIT:   LD      A,0
        CALL    GET.STATUS      ;when exit, you must select S#0
        EI
        RET

GET.STATUS:                     ;read status register specified by A
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET
```

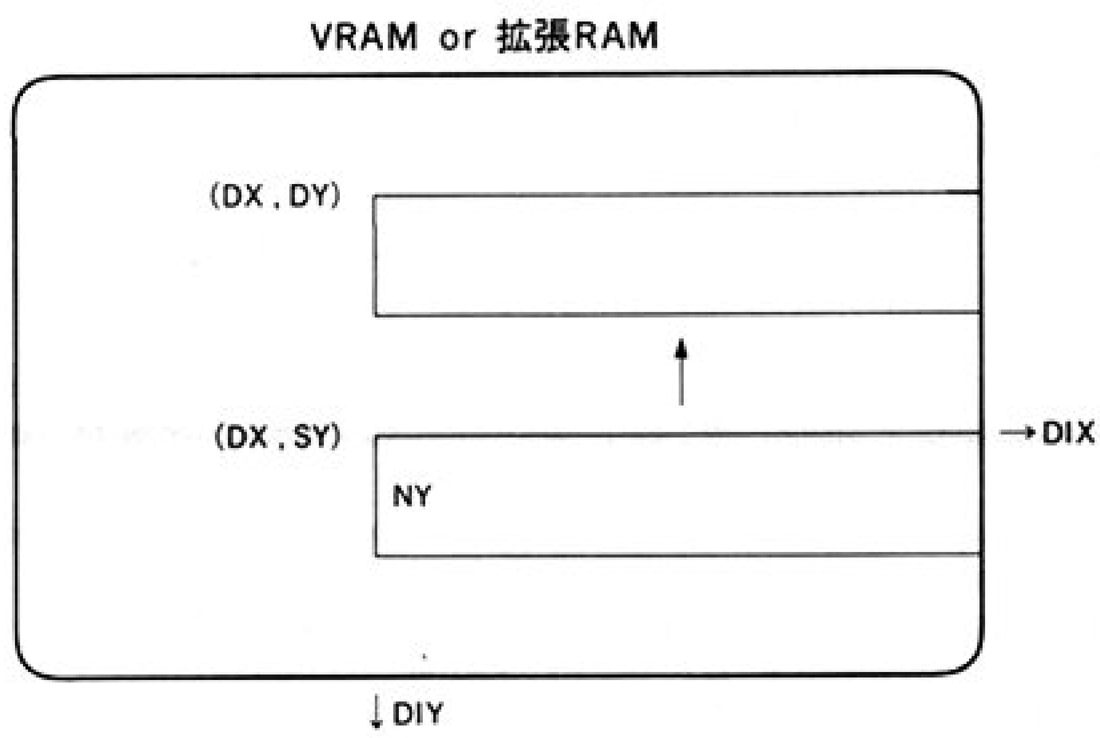
```
WAIT.VDP:                          ;wait VDP ready
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JR      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.2 YMMM(Y軸方向のVRAM間高速転送)

VRAMの指定された領域のデータをVRAMの他の領域にバイト単位で転送します.ただしこのコマンドでの転送はY軸方向だけに限られます(図4.78).

図4.79に示すデータをレジスタに設定した後, R#46にコマンドコードE0Hを書き込むとコマンドの実行が開始されます. S#2のCEビットが"1"の間はコマンド実行中であることを示します. **リスト4.9**にYMMMのサンプルを示します.



MXD : 転送先　メモリ選択　　0 = VRAM, 1 = 拡張RAM

SY　: 転送元基準点　Y座標　(0〜1023)

NY　: Y方向転送ドット数　(0〜1023)

DIX : 転送元基準点から左右どちらの画面端までを転送するのかを設定する. 0 = 右, 1 = 左

DIY : 転送基準点から見たNYの方向. 0 = 下, 1 = 上

DX　: 転送先基準点　X座標　(0〜511) ※

DY　: 転送先基準点　Y座標　(0〜1023)

※DXは, GRAPHIC 4/GRAPHIC 6モードのときは下位1ビット, GRAPHIC 5モードのときは下位2ビットが無視される.

図4.78　YMMMコマンドの動作

▶YMMMレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY→転送元基準点
R#35	0	0	0	0	0	0	SY9	SY8	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX→転送先および転送元基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY→転送先基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数
R#43	0	0	0	0	0	0	NY9	NY8	
R#45	0	−	MXD	−	DIY	DIX	−	−	ARG (Argument register)

DIX → 転送方向（X）
DIY → 転送方向（Y）
MXD → 転送先メモリ選択

▶YMMMコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	1	1	1	0	−	−	−	−	CMR

図4.79　YMMMコマンドのレジスタ設定

リスト4.9　YMMMコマンド実行例

```
;=============================================================
;  List 4.9    YMMM sample
;                 to use, set L,E,B,C,D(bit 2) and go
;                 VRAM (B,L)-(*,E) ---> VRAM (B,C)
;                 DIX must be set in D(bit 2)
;=============================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

YMMM:   DI                      ;disable interrupt
        PUSH    BC              ;save distination
        CALL    WAIT.VDP        ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C               ;C := PORT#1's address
        LD      A,34
        OUT     (C),A
        LD      A,17+80H
        OUT     (C),A           ;R#17 := 34
```

```
        INC     C
        INC     C               ;C := PORT#3's address
        XOR     A
        OUT     (C),L           ;SY
        OUT     (C),A

        LD      A,L             ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,YMMM1
        LD      E,00000000B
        NEG
YMMM1:  LD      L,A             ;L := NY , D := DIY

        LD      A,D
        OR      E

        POP     DE              ;restore DX,DY
        PUSH    AF              ;save DIX,DIY
        XOR     A
        OUT     (C),D           ;DX
        OUT     (C),A
        OUT     (C),E           ;DY
        OUT     (C),A
        OUT     (C),A           ;dummy
        OUT     (C),A           ;dummy
        OUT     (C),L           ;NY
        OUT     (C),A
        OUT     (C),A           ;dummy
        POP     AF
        OUT     (C),A           ;DIX and DIY
        LD      A,11010000B     ;YMMM command
        OUT     (C),A

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

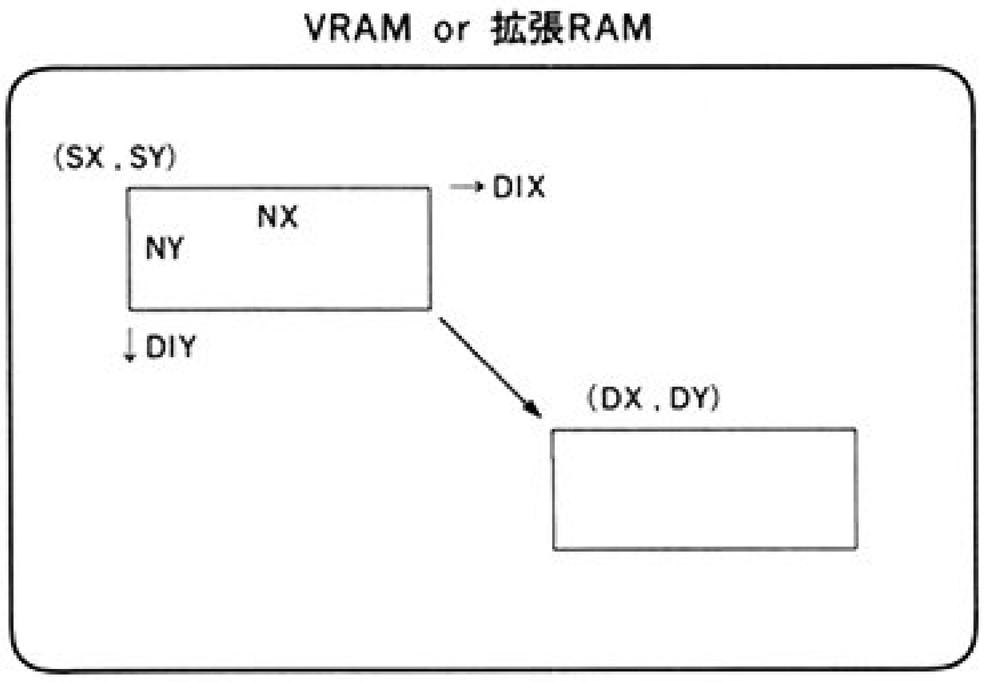
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.3 HMMM(VRAM 間高速転送)

VRAM の指定された領域のデータを VRAM の他の領域にバイト単位で転送します(**図 4.80**).

図 4.81 に示すパラメータをセットした後，R#46に D0H を書き込むとコマンドを実行します．コマンド実行中は，S#2の CE ビットが“1”になります．**リスト 4.10** に HMMM のサンプルを示します．



MXS：転送元　メモリ選択　　0＝VRAM，1＝拡張RAM
MXD：転送先　メモリ選択　　0＝VRAM，1＝拡張RAM

SX　：転送元基準点　X 座標　(0〜511) ※
SY　：転送元基準点　Y 座標　(0〜1023)

NX　：X 方向転送ドット数　(0〜511) ※
NY　：Y 方向転送ドット数　(0〜1023)

DIX：基準点からのNXの方向　　0＝右，1＝左
DIY：基準点からのNYの方向　　0＝下，1＝上

DX　：転送先基準点　X 座標　(0〜511) ※
DY　：転送先基準点　Y 座標　(0〜1023)

※SX，DX，NXともに，GRAPHIC 4／GRAPHIC 6モードのときは下位1ビット，GRAPHIC 5モードのときは下位2ビットが無視される．

図4.80　HMMMコマンドの動作

▶HMMMレジスタセットアップ

MSB	7	6	5	4	3	2	1	0	LSB	
R#32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	SX	転送元基準点
R#33	0	0	0	0	0	0	0	SX8		
R#34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY	
R#35	0	0	0	0	0	0	SY9	SY8		
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX	転送先基準点
R#37	0	0	0	0	0	0	0	DX8		
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY	
R#39	0	0	0	0	0	0	DY9	DY8		
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向転送ドット数	
R#41	0	0	0	0	0	0	0	NX8		
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数	
R#43	0	0	0	0	0	0	NY9	NY8		

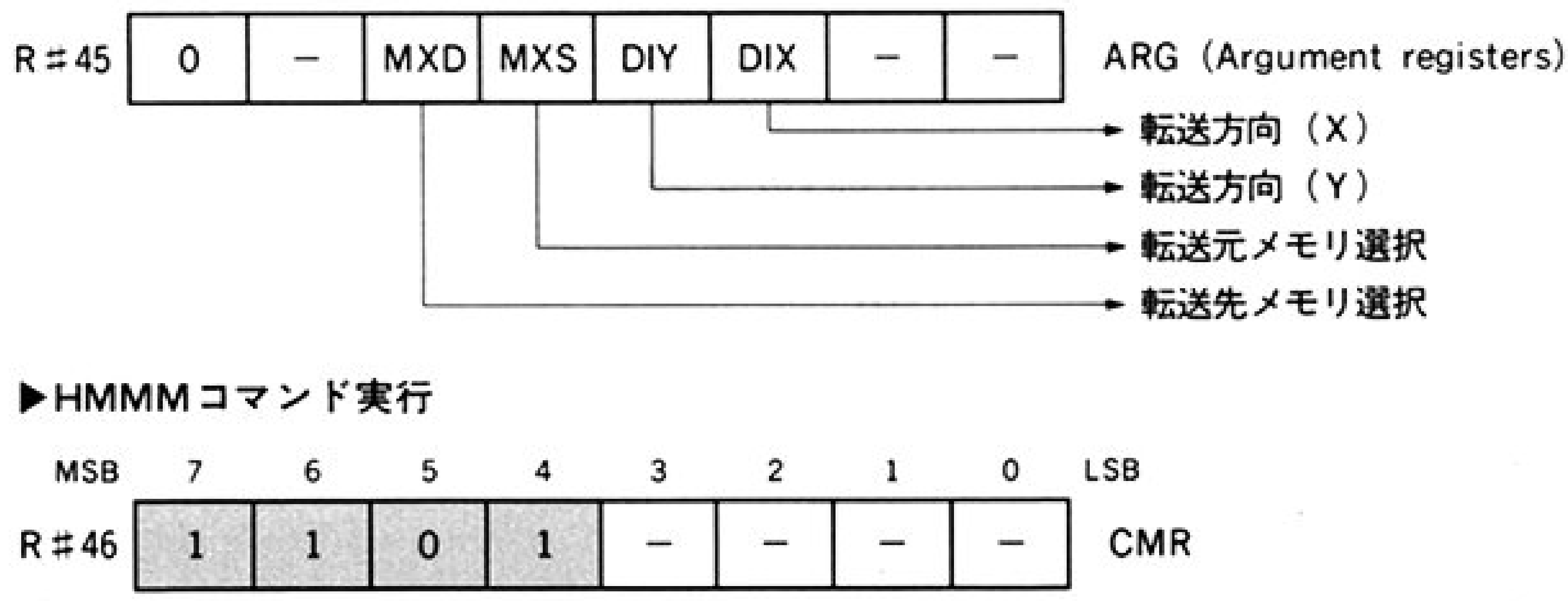


図4.81　HMMMコマンドのレジスタ設定

リスト4.10　HMMMコマンド実行例

```
;==========================================================
;  List 4.10    HMMM sample
;                 to use, set H,L,D,E,B,C and go
;                 VRAM (H,L)-(D,E) ---> VRAM (B,C)
;==========================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

HMMM:   DI                              ;disable interrupt
        PUSH    BC                      ;save distination (DX,DY)
        CALL    WAIT.VDP                ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C                       ;C := PORT#'s address
        LD      A,32
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A                   ;R#17 := 32

        INC     C
        INC     C                       ;C := PORT#3's address
        XOR     A
        OUT     (C),H                   ;SX
        OUT     (C),A
        OUT     (C),L                   ;SY
        OUT     (C),A

        LD      A,H                     ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,HMMM1
        LD      D,00000000B
        NEG
HMMM1:  LD      H,A                     ;H := NX , D := DIX

        LD      A,L                     ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,HMMM2
        LD      E,00000000B
        NEG
HMMM2:  LD      L,A                     ;L := NY , E := DIY
```

```
        LD      A,D
        OR      E
        POP     DE              ;restore DX,DY
        PUSH    AF              ;save DIX,DIY
        XOR     A
        OUT     (C),D           ;DX
        OUT     (C),A
        OUT     (C),E           ;DY
        OUT     (C),A
        OUT     (C),H           ;NX
        OUT     (C),A
        OUT     (C),L           ;NY
        OUT     (C),A
        OUT     (C),A           ;dummy
        POP     AF
        OUT     (C),A           ;DIX and DIY

        LD      A,11010000B     ;HMMM command
        OUT     (C),A

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

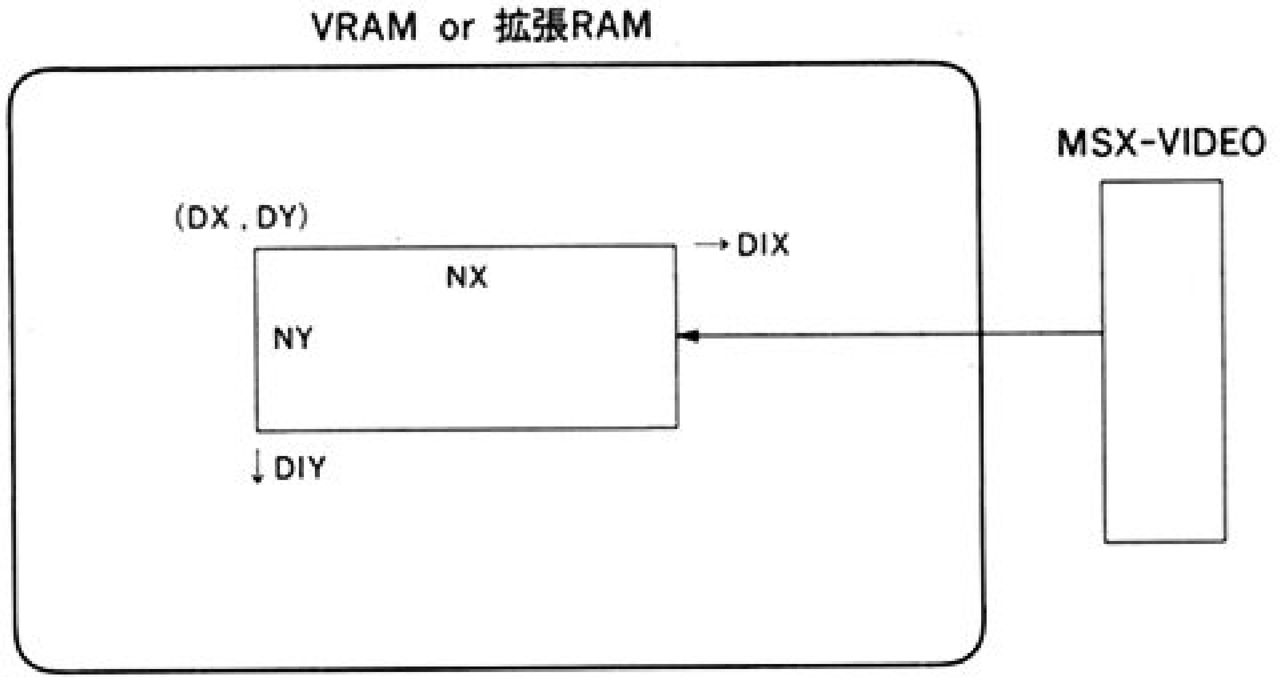
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.4 HMMV(長方形の高速塗りつぶし)

VRAMの指定された領域をバイト単位の色コードデータで塗りつぶします(**図4.82**).

図4.83に示したパラメータをセットした後，R#46にC0Hを書き込むとコマンドを実行します．コマンド実行中はS#2のCEビットが1になります．**リスト4.11**にHMMVのサンプルを示します．



MXD : メモリ選択　　0＝VRAM，1＝拡張RAM

NX　: X方向塗りつぶしドット数　(0～511) ※
NY　: Y方向塗りつぶしドット数　(0～1023)

DIX　: 基準点からのNXの方向　　0＝右，1＝左
DIY　: 基準点からのNYの方向　　0＝下，1＝上

DX　: 基準点　X座標　(0～511) ※
DY　: 基準点　Y座標　(0～1023)

CLR(R#44 : Color register) : 塗りつぶしデータ

※DX，NXともに，GRAPHIC 4/GRAPHIC 6モードのときは下位1ビット，GRAPHIC 5モードのときは下位2ビットが無視される．

図4.82　HMMVコマンドの動作

▶HMMVレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX } 基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY } 基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向塗りつぶしドット数
R#41	0	0	0	0	0	0	0	NX8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向塗りつぶしドット数
R#43	0	0	0	0	0	0	NY9	NY8	

図4.83　HMMVコマンドのレジスタ設定

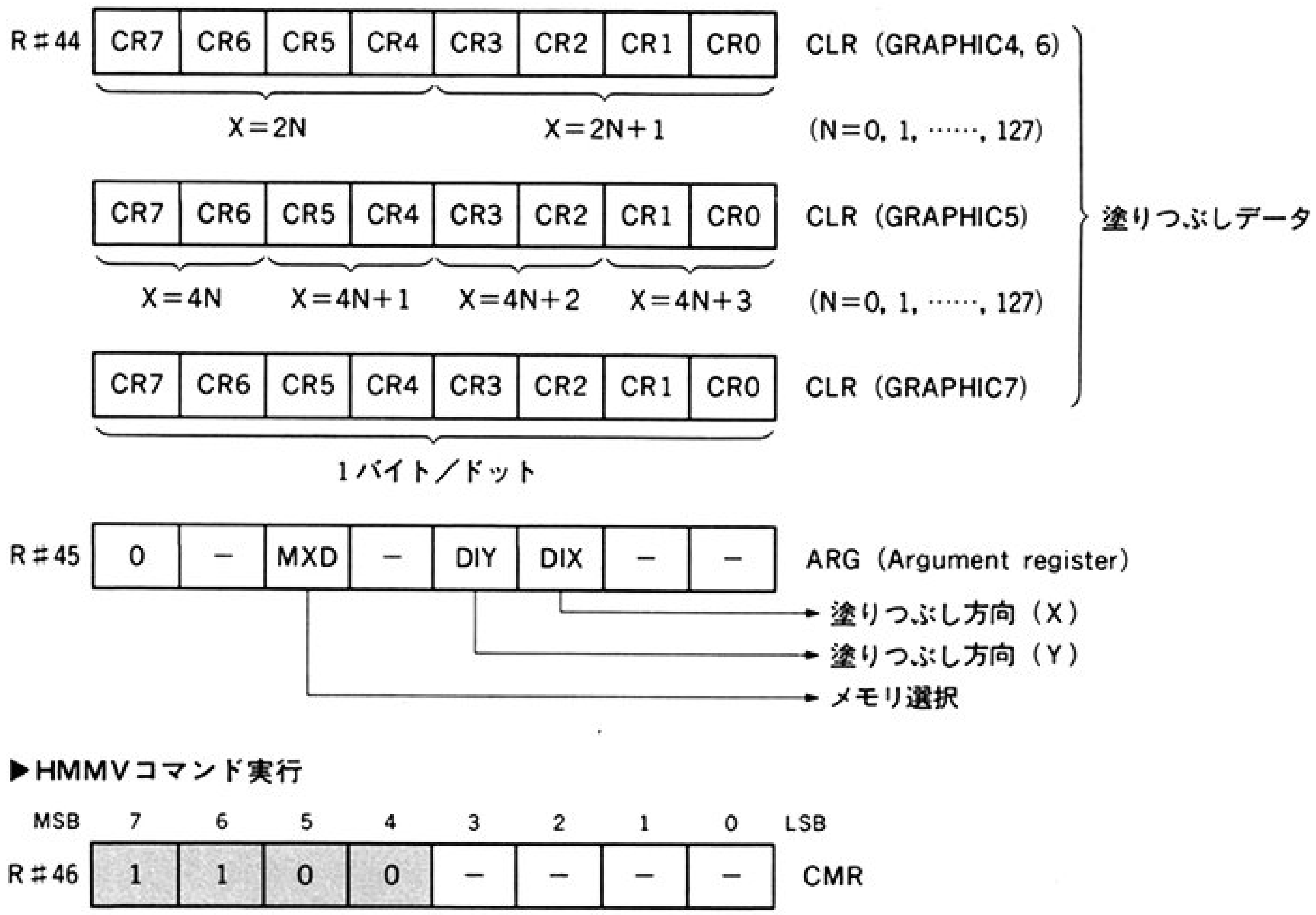


図4.83(続き)　HMMVコマンドのレジスタ設定

リスト4.11　HMMVコマンド実行例

```
;==================================================
;  List 4.11    HMMV sample
;                 to use, set H,L,D,E,B and go
;                 B ---> VRAM (H,L)-(D,E) fill
;==================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

HMMV:   DI                          ;disable interrupt
        CALL    WAIT.VDP            ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C                   ;C := PORT#1's address
        LD      A,36
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A               ;R#17 := 36

        INC     C
        INC     C                   ;C := PORT#1's address
        XOR     A
        OUT     (C),H               ;DX
        OUT     (C),A
        OUT     (C),L               ;DY
        OUT     (C),A
```

```
        LD      A,H             ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,HMMV1
        LD      D,00000000B
        NEG
HMMV1:  LD      H,A             ;H := NX

        LD      A,L             ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,HMMV2
        LD      E,00000000B
        NEG
HMMV2:  OUT     (C),H
        LD      H,A             ;H := NY

        XOR     A
        OUT     (C),A
        OUT     (C),H
        OUT     (C),A
        OUT     (C),B           ;fill data
        XOR     A
        OR      D
        OR      E
        OUT     (C),A           ;DIX and DIY

        LD      A,11000000B     ;HMMV command
        OUT     (C),A

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

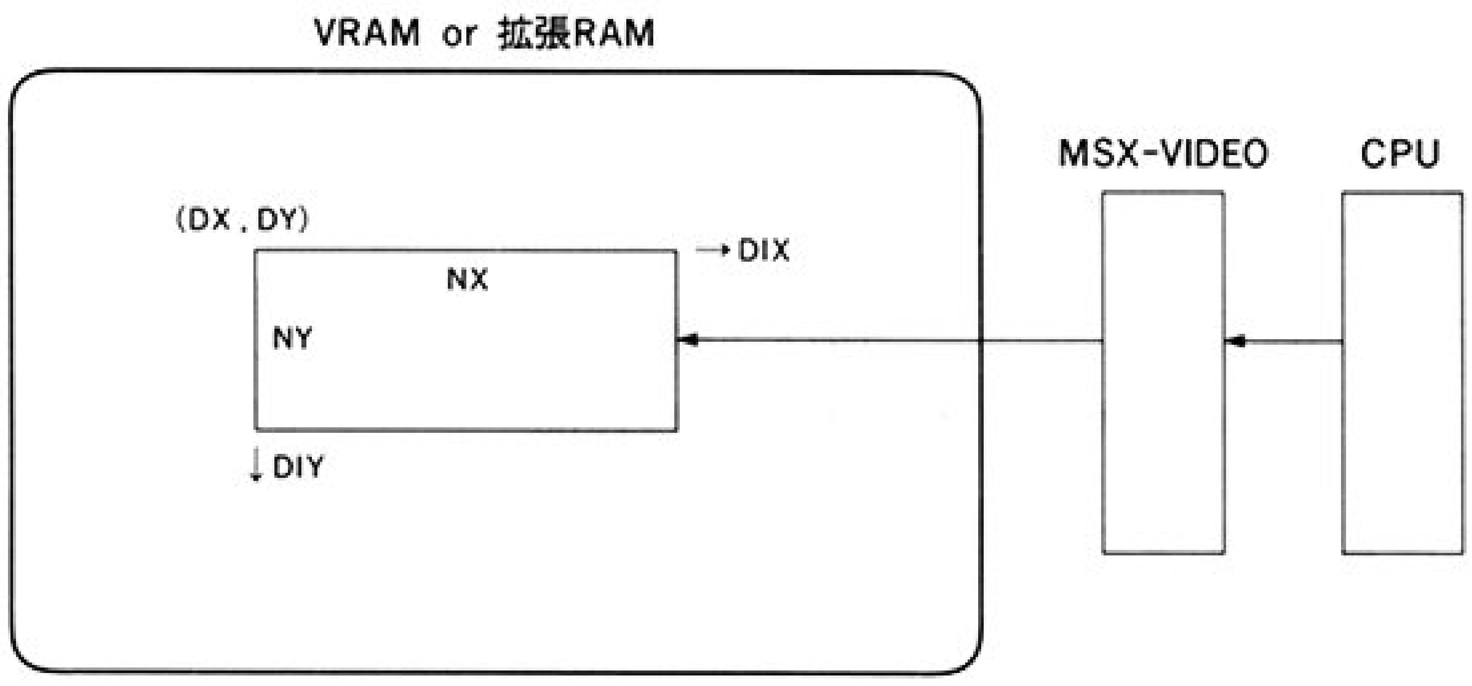
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.5 LMMC(CPU → VRAM 論理転送)

CPUからVRAMの指定された領域にデータをドット単位で転送します(図4.84).このとき転送先とのロジカルオペレーションを指定することが可能です.なお,このLMMCを始めとする論理転送コマンドではデータはドット単位で転送され，どの画面モードにおいても1ドットの情報につき1バイトを必要とします.

図4.85のデータをセットした後，R#46にコマンドコードB0Hを書き込みます．また，その時コマンドレジスタの下位4ビットを用いて，ロジカルオペレーションの指定が可能です．データの転送方法はHMMCと同様，S#2のTRビットとCEビットを参照しながら行います(図4.86)．リスト4.12にLMMCの例を示します.



MXD：転送先　メモリ選択　　0＝VRAM，1＝拡張RAM

NX　：X方向転送ドット数　(0〜511)
NY　：Y方向転送ドット数　(0〜1023)

DIX　：基準点からのNXの方向　　0＝右，1＝左
DIY　：基準点からのNYの方向　　0＝下，1＝上

DX　：転送先基準点　X座標　(0〜511)
DY　：転送先基準点　Y座標　(0〜1023)

CLR(R#44 : Color register)：転送データの第1バイト

図4.84　LMMCコマンドの動作

▶LMMCレジスタセットアップ

MSB	7	6	5	4	3	2	1	0	LSB
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX (転送先基準点)
R#37	0	0	0	0	0	0	0	DX8	DX (転送先基準点)
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY (転送先基準点)
R#39	0	0	0	0	0	0	DY9	DY8	DY (転送先基準点)

R♯40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向転送ドット数
R♯41	0	0	0	0	0	0	0	NX8	
R♯42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数
R♯43	0	0	0	0	0	0	NY9	NY8	
R♯44	−	−	−	−	CR3	CR2	CR1	CR0	CLR (GRAPHIC4, 6) 転送データ
	−	−	−	−	−	−	CR1	CR0	CLR (GRAPHIC5) 転送データ
	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	CLR (GRAPHIC7) 転送データ
R♯45	0	−	MXD	−	DIY	DIX	−	−	ARG (Argument register)

DIX → 転送方向（X）
DIY → 転送方向（Y）
MXD → 転送先メモリ選択

▶LMMCコマンド実行

MSB	7	6	5	4	3	2	1	0	LSB
R♯46	1	0	1	1	LO3	LO2	LO1	LO0	CMR

LO3〜LO0 → ロジカルオペレーション

図4.85　LMMCコマンドのレジスタ設定

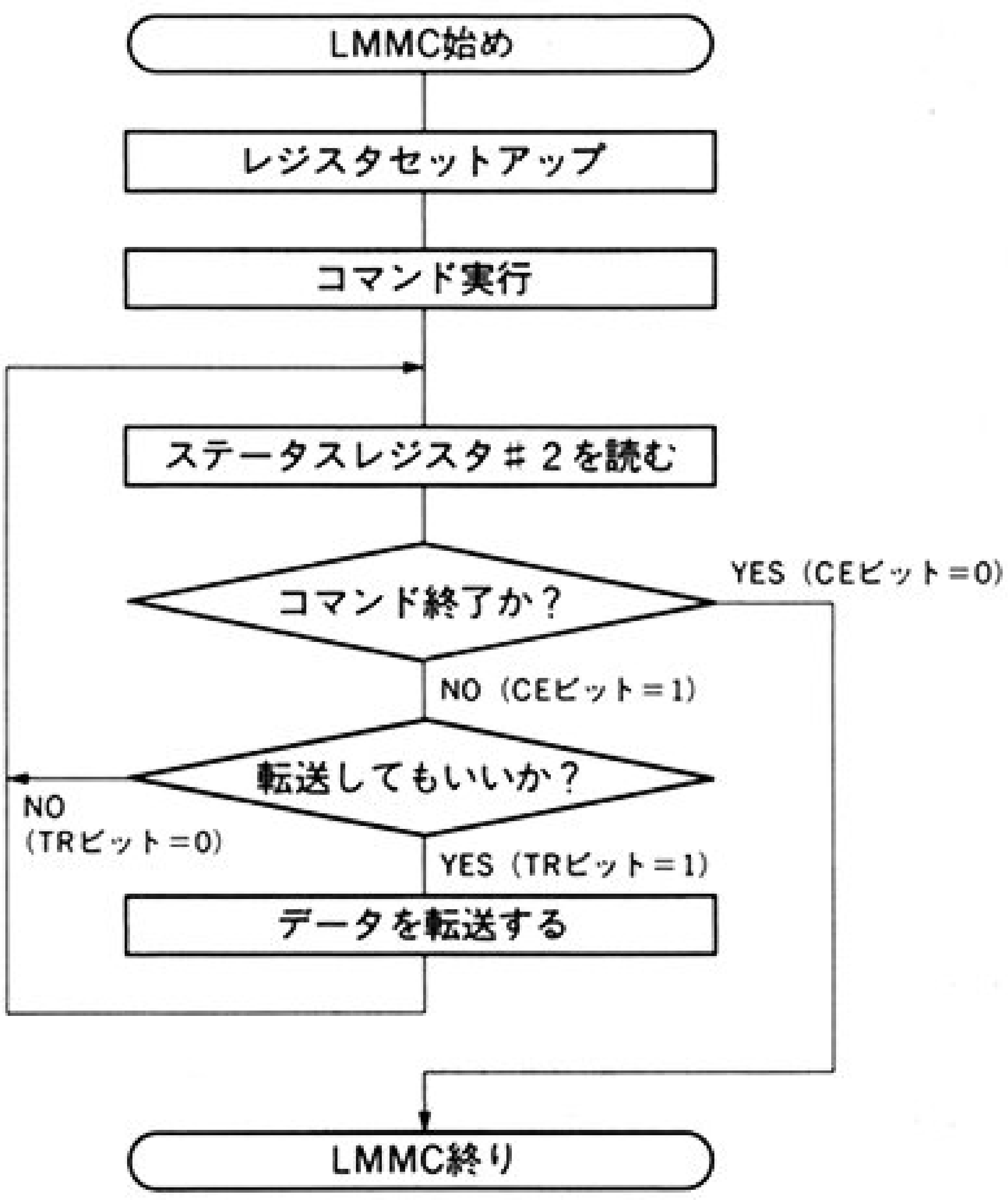


図4.86　LMMCコマンド実行のフローチャート

リスト4.12　LMMCコマンドの実行例

```
;=============================================================
;  List 4.12   LMMC sample
;               to use, set H,L,D,E,IX,A and go
;               RAM [IX] ---> VRAM (H,L)-(D,E) (logi-OP : A)
;=============================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

LMMC:   DI                              ;disable interrupt
        LD      B,A                     ;B := LOGICAL OPERATION
        CALL    WAIT.VDP                ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C                       ;C := PORT#1's address
        LD      A,36
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A                   ;R#17 := 36

        INC     C
        INC     C                       ;C := PORT#3's address
        XOR     A
        OUT     (C),H                   ;DX
        OUT     (C),A
        OUT     (C),L                   ;DY
        OUT     (C),A

        LD      A,H                     ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,LMMC1
        LD      D,00000000B
        NEG
LMMC1:  LD      H,A                     ;H := NX , D := DIX

        LD      A,L                     ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,LMMC2
        LD      E,00000000B
        NEG
LMMC2:  LD      L,A                     ;L := NY , E := DIY

        XOR     A
        OUT     (C),H                   ;NX
        OUT     (C),A
        OUT     (C),L                   ;NY
        OUT     (C),A
        LD      A,(IX+0)
        OUT     (C),A                   ;first DATA
        LD      A,D
        OR      E
        OUT     (C),A                   ;DIX and DIY

        LD      A,B                     ;A := LOGICAL OPERATION
        OR      10110000B               ;LMMC command
        OUT     (C),A
```

```
        DEC     C
        DEC     C               ;C := PORT#1's address
LOOP:   LD      A,2
        CALL    GET.STATUS
        BIT     0,A             ;check CE bit
        JP      Z,EXIT
        BIT     7,A             ;check TR bit
        JP      Z,LOOP
        INC     IX
        LD      A,(IX+0)
        OUT     (C),A
        JP      LOOP

EXIT:   LD      A,0
        CALL    GET.STATUS

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

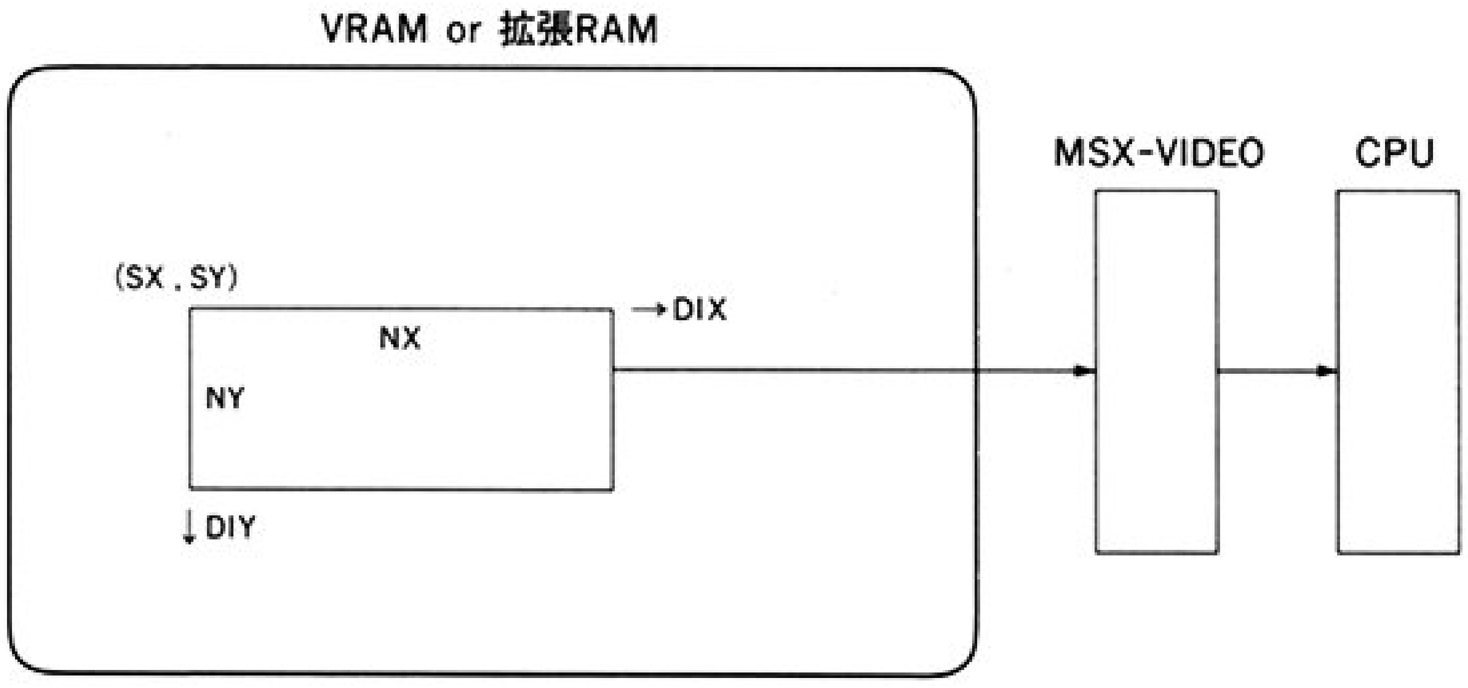
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.6　LMCM(VRAM → CPU 論理転送)

VRAMの指定された領域からCPUにデータをドット単位で転送します(**図4.87**).

図4.88に示したパラメータをセットした後，R#46にコマンドコードA0Hを書き込むとコマンドの実行が開始され，MSX-VIDEOからデータが転送され始めます．CPU側では，まずS#2のTRビットを参照し，この値が“1”ならばMSX-VIDEOのデータが用意できたことを示すので，S#7からデータを読み出します．S#2のCEビットが“0”になれば，データが終了したことを示します(**図4.89**)．**リスト4.13**にLMCMの例を示します．



MXS：転送元　メモリ選択　　0＝VRAM，1＝拡張RAM

SX　：転送元基準点　X座標　(0～511)
SY　：転送元基準点　Y座標　(0～1023)

NX　：X方向転送ドット数　(0～511)
NY　：Y方向転送ドット数　(0～1023)

DIX　：基準点からのNXの方向　　0＝右，1＝左
DIY　：基準点からのNYの方向　　0＝下，1＝上

図4.87　LMCMコマンドの動作

▶LMCMレジスタセットアップ

MSB	7	6	5	4	3	2	1	0	LSB
R#32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	SX } 転送元基準点
R#33	0	0	0	0	0	0	0	SX8	
R#34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY } 転送元基準点
R#35	0	0	0	0	0	0	SY9	SY8	
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向転送ドット数
R#41	0	0	0	0	0	0	0	NX8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数
R#43	0	0	0	0	0	0	NY9	NY8	
R#45	0	−	−	MXS	DIY	DIX	−	−	ARG (Argument registers)

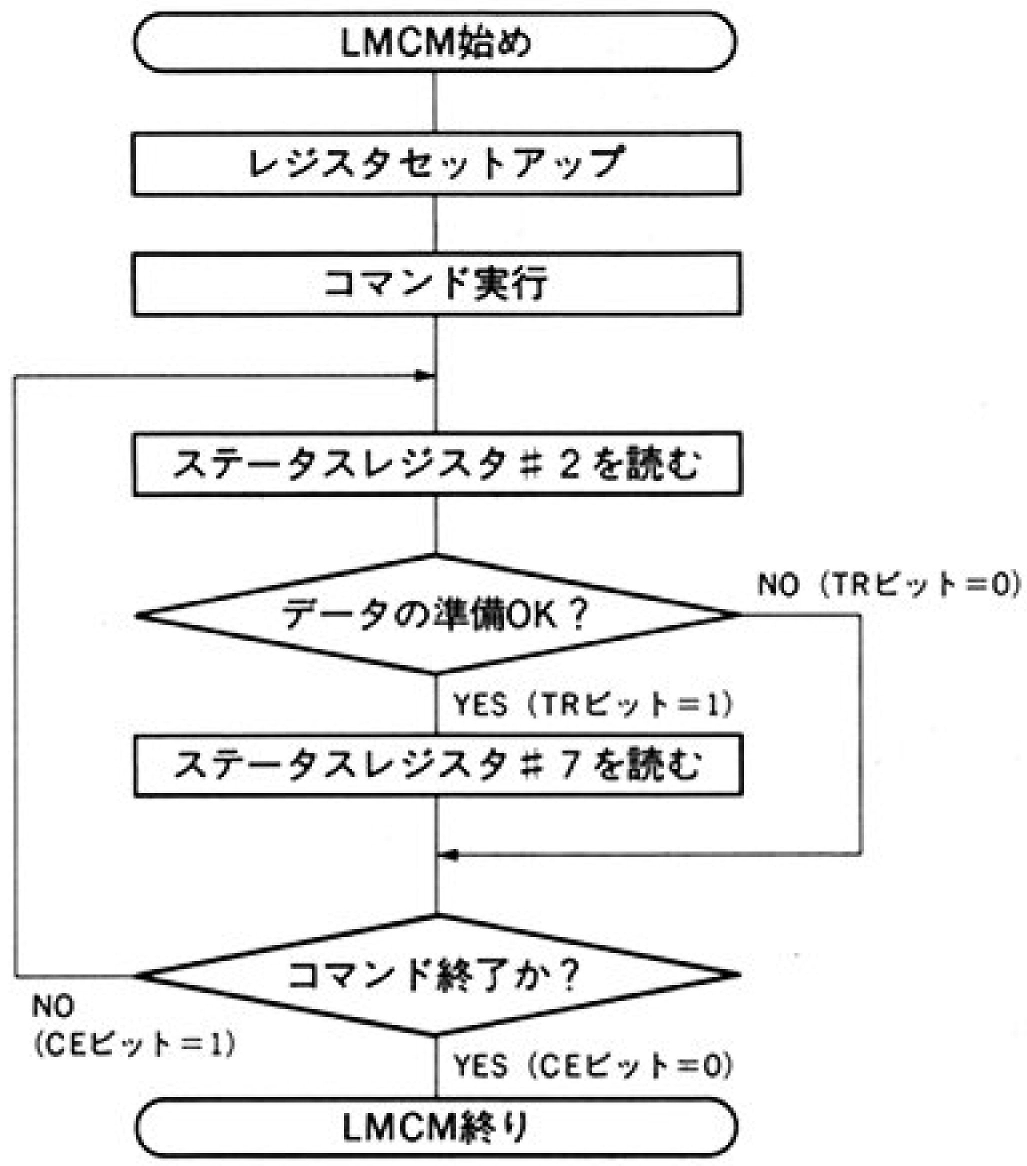
DIX → 転送方向（X）
DIY → 転送方向（Y）
MXS → 転送元メモリ選択

▶LMCMコマンド実行

MSB	7	6	5	4	3	2	1	0	LSB
R#46	1	0	1	0	−	−	−	−	CMR

S#7	0	0	0	0	C3	C2	C1	C0	ステータスレジスタ（GRAPHIC4, 6）
S#7	0	0	0	0	0	0	C1	C0	ステータスレジスタ（GRAPHIC5）
S#7	C7	C6	C5	C4	C3	C2	C1	C0	ステータスレジスタ（GRAPHIC7）

図4.88　LMCMコマンドのレジスタ設定



注1．コマンド実行の前にTRビットをリセットする必要があるので，"レジスタセットアップ"の中でステータスレジスタS#7をリードしてください．
　2．最後のデータがステータスレジスタS#7にセットされて，TRビット＝1となっていても，MSX-VIDEO内部ではコマンドは終了し，CE＝0となります．

図4.89　LMCMコマンド実行のフローチャート

リスト4.13　LMCMコマンドの実行例

```
;=============================================================
;  List 4.13    LMCM sample
;                   to use, set H,L,D,E,IX,A and go
;                   VRAM (H,L)-(D,E) ---> RAM [IX]
;=============================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

LMMC:   DI                              ;disable interrupt
        LD      B,A                     ;B := LOGICAL OPERATION
        CALL    WAIT.VDP                ;wait end of command

        LD      A,(WRVDP)
```

```
        LD      C,A
        INC     C               ;C := PORT#1's address
        LD      A,32
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A           ;R#17 := 32
        INC     C
        INC     C               ;C := PORT#3's address
        XOR     A
        OUT     (C),H           ;SX
        OUT     (C),A
        OUT     (C),L           ;SY
        OUT     (C),A
        OUT     (C),A           ;dummy
        OUT     (C),A           ;dummy
        OUT     (C),A           ;dummy
        OUT     (C),A           ;dummy
        LD      A,H             ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,LMMC1
        LD      D,00000000B
        NEG
LMMC1:  LD      H,A             ;H := NX , D := DIX

        LD      A,L             ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,LMMC2
        LD      E,00000000B
        NEG
LMMC2:  LD      L,A             ;L := NY , E := DIY

        XOR     A
        OUT     (C),H           ;NX
        OUT     (C),A
        OUT     (C),L           ;NY
        OUT     (C),A
        LD      A,(IX+0)
        OUT     (C),A           ;dummy
        LD      A,D
        OR      E
        OUT     (C),A           ;DIX and DIY
        LD      A,7
        CALL    GET.STATUS      ;TR bit reset
        LD      A,B             ;A := LOGICAL OPERATION
        OR      10100000B       ;LMMC command
        OUT     (C),A
        LD      A,(RDVDP)
        LD      C,A             ;C := PORT#1's address
LOOP:   LD      A,2
        CALL    GET.STATUS
        BIT     0,A             ;check CE bit
        JP      Z,EXIT
        BIT     7,A             ;check TR bit
        JP      Z,LOOP
        LD      A,7
        CALL    GET.STATUS
        LD      (IX+0),A
        INC     IX
        JP      LOOP
EXIT:   LD      A,0
        CALL    GET.STATUS
        EI
        RET
GET.STATUS:
        PUSH    BC
```

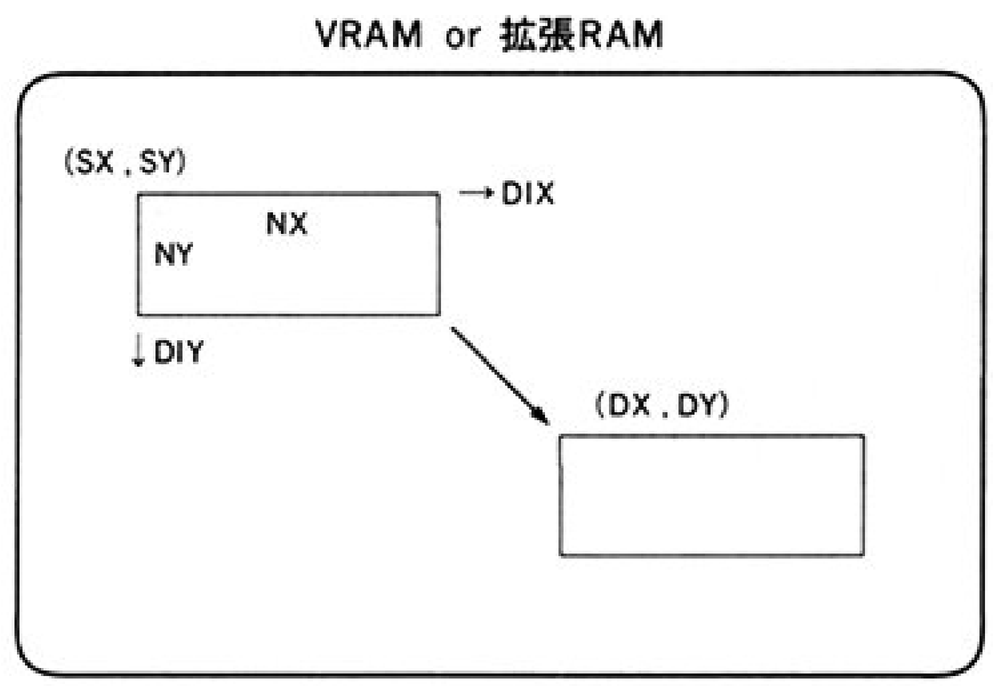
```
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.7 LMMM(VRAM → VRAM 論理転送)

VRAMの指定された領域のデータをVRAMの他の領域にドット単位で転送します(**図 4.90**). **図 4.91**に示すパラメータをセットした後，R#46にコマンドコード9XH(Xはロジカルオペレーション)を書き込むとコマンドが実行されます．S#2のCEビットが"1"の間はコマンド実行中であることを示します．**リスト 4.14**にLMMMの例を示します．



MXS：転送元メモリ選択　0＝VRAM，1＝拡張RAM
MXD：転送先メモリ選択　0＝VRAM，1＝拡張RAM

SX　：転送元基準点　X座標　(0〜511)
SY　：転送元基準点　Y座標　(0〜1023)

NX　：X方向転送ドット数　(0〜511)
NY　：Y方向転送ドット数　(0〜1023)

DIX　：基準点からのNXの方向　0＝右，1＝左
DIY　：基準点からのNYの方向　0＝下，1＝上

DX　：転送先基準点　X座標　(0〜511)
DY　：転送先基準点　Y座標　(0〜1023)

図4.90　LMMMコマンドの動作

▶LMMMレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	SX 転送元基準点
R#33	0	0	0	0	0	0	0	SX8	
R#34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY 転送元基準点
R#35	0	0	0	0	0	0	SY9	SY8	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX 転送先基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY 転送先基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向転送ドット数
R#41	0	0	0	0	0	0	0	NX8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向転送ドット数
R#43	0	0	0	0	0	0	NY9	NY8	
R#45	0	−	MXD	MXS	DIY	DIX	−	−	ARG (Argument registers)

DIX → 転送方向（X）
DIY → 転送方向（Y）
MXS → 転送元メモリ選択
MXD → 転送先メモリ選択

▶LMMMコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	1	0	0	1	LO3	LO2	LO1	LO0	CMR

LO3〜LO0 → ロジカルオペレーション

図4.91　LMMMコマンドのレジスタ設定

リスト4.14　LMMMコマンドの実行例

```
;=============================================================
;  List 4.14    LMMM sample
;                  to use, set H,L,D,E,B,C,A and go
;                  VRAM (H,L)-(D,E) ---> VRAM (B,C) (logi-OP : A)
;=============================================================
;
RDVDP    EQU      0006H
WRVDP    EQU      0007H
```

```
;----- program start -----

LMMM:   DI                              ;disable interrupt
        PUSH    AF                      ;save LOGICAL OPERATION
        PUSH    BC                      ;save DESTINATION
        CALL    WAIT.VDP                ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C                       ;C := PORT#1's address
        LD      A,32
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A                   ;R#17 := 32

        INC     C
        INC     C                       ;C := PORT#3's address
        XOR     A
        OUT     (C),H                   ;SX
        OUT     (C),A
        OUT     (C),L                   ;SY
        OUT     (C),A

        LD      A,H                     ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,LMMM1
        LD      D,00000000B
        NEG
LMMM1:  LD      H,A                     ;H := NX , D := DIX

        LD      A,L                     ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,LMMM2
        LD      E,00000000B
        NEG
LMMM2:  LD      L,A                     ;L := NY , E := DIY

        LD      A,D
        OR      E
        POP     DE                      ;restore DX,DY
        PUSH    AF                      ;save DIX,DIY
        XOR     A
        OUT     (C),D                   ;DX
        OUT     (C),A
        OUT     (C),E                   ;DY
        OUT     (C),A
        OUT     (C),H                   ;NX
        OUT     (C),A
        OUT     (C),L                   ;NY
        OUT     (C),A
        OUT     (C),A                   ;dummy
        POP     AF
        OUT     (C),A                   ;DIX AND DIY

        POP     AF                      ;A := LOGICAL OPERATION
        OR      10010000B               ;LMMM command
        OUT     (C),A

        EI
        RET
```

```
GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

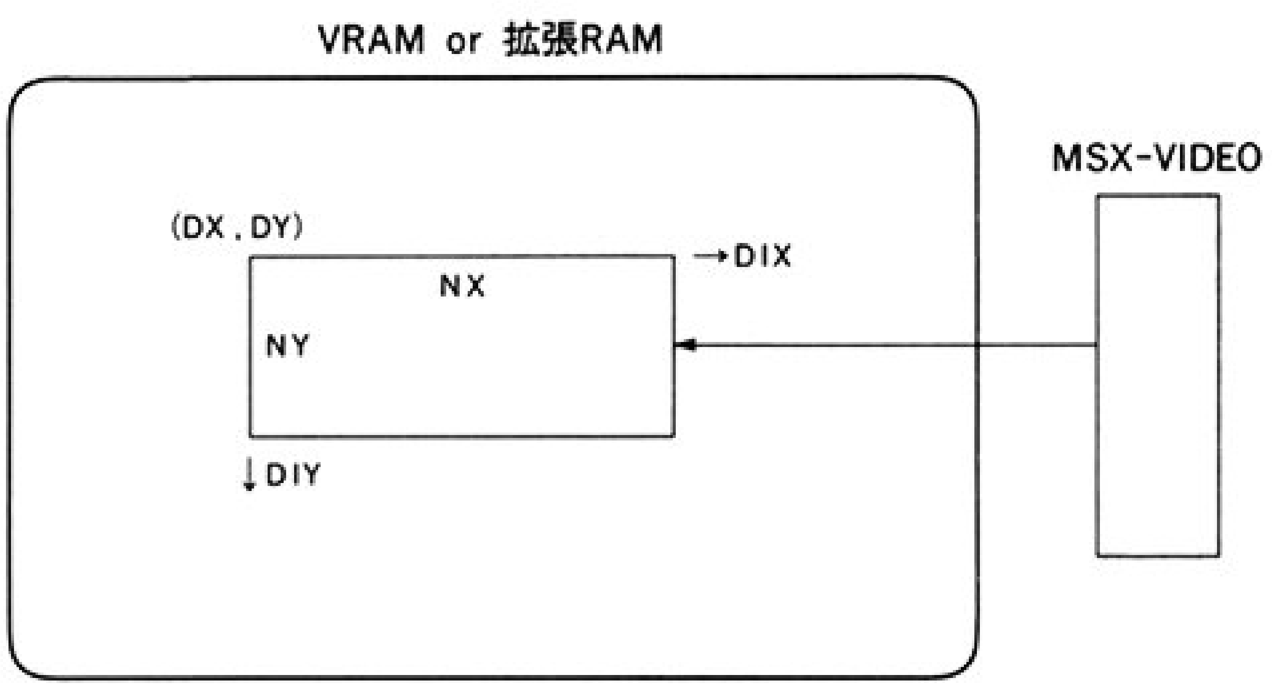
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.8　LMMV(VRAM論理塗りつぶし)

VRAMの指定された領域をドット単位の色コードで塗りつぶします(図4.92)。この時VRAM上のデータと指定したデータの間でロジカルオペレーションを行うことができます。

図4.93に示すパラメータをセットした後，R#46にコマンドコード8XH(Xはロジカルオペレーション)を書き込むとコマンドが実行されます．S#2のCEビットが"1"の間はコマンド実行中であることを示します．**リスト4.15**にLMMVの例を示します．



MXD：メモリ選択　　0＝VRAM，1＝拡張RAM

NX　：X方向塗りつぶしドット数　(0〜511)
NY　：Y方向塗りつぶしドット数　(0〜1023)

DIX　：基準点からのNXの方向　　0＝右，1＝左
DIY　：基準点からのNYの方向　　0＝下，1＝上

DX　：基準点　X座標　(0〜511)
DY　：基準点　Y座標　(0〜1023)

CLR(R#44：Color register)：塗りつぶしデータ

図4.92　LMMVコマンドの動作

▶**LMMVレジスタセットアップ**

	MSB 7	6	5	4	3	2	1	0 LSB	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX } 基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY } 基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	NX→X方向塗りつぶしドット数
R#41	0	0	0	0	0	0	0	NX8	
R#42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	NY→Y方向塗りつぶしドット数
R#43	0	0	0	0	0	0	NY9	NY8	
R#44	0	0	0	0	C3	C2	C1	C0	CLR (GRAPHIC4, 6) } 転送データ
	0	0	0	0	0	0	C1	C0	CLR (GRAPHIC5) } 転送データ
	C7	C6	C5	C4	C3	C2	C1	C0	CLR (GRAPHIC7) } 転送データ
R#45	−	−	MXD	−	DIY	DIX	−	−	ARG (Argument register)

- DIX → 塗りつぶし方向 (X)
- DIY → 塗りつぶし方向 (Y)
- MXD → メモリ選択

▶**LMMVコマンド実行**

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	1	0	0	0	LO3	LO2	LO1	LO0	CMR

- LO3〜LO0 → ロジカルオペレーション

図4.93 LMMVコマンドのレジスタ設定

リスト4.15 LMMVコマンドの実行例

```
;==========================================================
;   List 4.15    LMMV sample
;                  to use, set H,L,D,E,B,A and go
;                  data B ---> fill VRAM (H,L)-(D,E) (logi-OP : A)
;==========================================================
;
RDVDP    EQU      0006H
WRVDP    EQU      0007H
```

```
;----- program satrt -----

LMMV:   DI                      ;disable interrupt
        PUSH    AF              ;save LOGICAL OPERATION
        PUSH    BC              ;save FILL DATA
        CALL    WAIT.VDP        ;wait end of command

        LD      A,(WRVDP)
        LD      C,A
        INC     C               ;C := PORT#1's address
        LD      A,36
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A           ;R#17 := 36

        INC     C
        INC     C               ;C := PORT#3's address
        XOR     A
        OUT     (C),H           ;DX
        OUT     (C),A
        OUT     (C),L           ;DY
        OUT     (C),A

        LD      A,H             ;make NX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,LMMV1
        LD      D,00000000B
        NEG
LMMV1:  LD      H,A             ;H := NX , D := DIX

        LD      A,L             ;make NY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,LMMV2
        LD      E,00000000B
        NEG
LMMV2:  LD      L,A             ;L := NY , E := DIY

        XOR     A
        OUT     (C),H           ;NX
        OUT     (C),A
        OUT     (C),H           ;NY
        OUT     (C),A
        POP     AF
        OUT     (C),A           ;FILL DATA
        LD      A,D
        OR      E
        OUT     (C),A           ;DIX and DIY

        POP     AF              ;restore LOGICAL OPERATION
        OR      10000000B       ;LMMV command
        OUT     (C),A

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
```

```
        POP     BC
        RET

WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

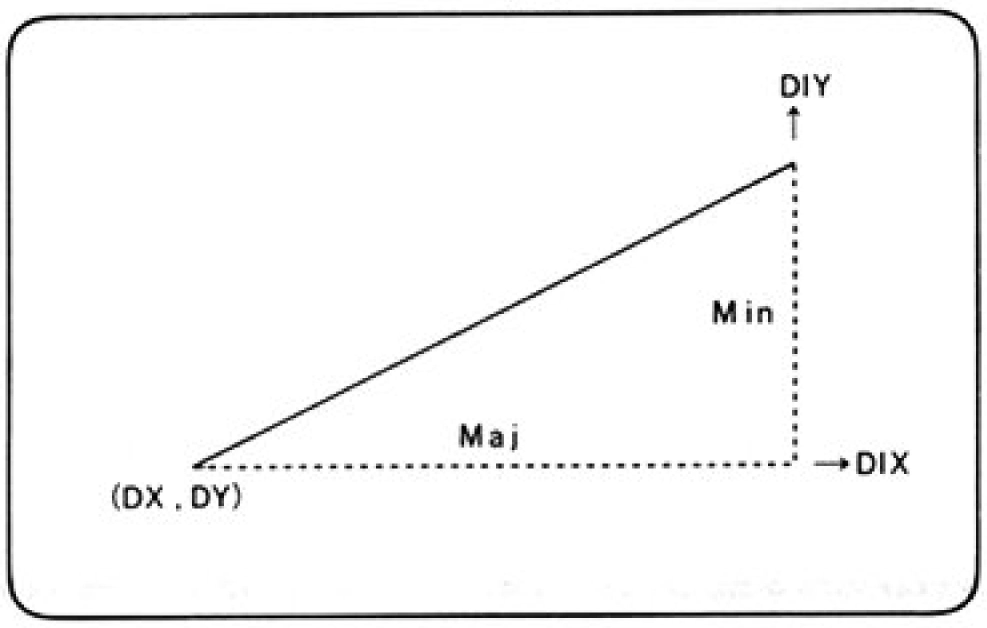
        END
```

6.5.9　LINE(直線の描画)

VRAM上の任意の座標に直線を描きます．指定するパラメータは基準点の座標と直線の伸びる方向，そして直線を対角線とする長方形の長辺と短辺の長さです(図4.94)．MajとMinを設定する時は，その前にかならず両辺の比較を行って，長い方をMajに設定してください．

図4.95に示すパラメータを設定した後，R#46に7XH(Xはロジカルオペレーション)を書き込むとコマンドが実行されます．S#2のCEビットが1の間はコマンド実行中であることを示します．リスト4.16にLINEの例を示します．

VRAM or 拡張RAM



MXD：メモリ選択　　0＝VRAM，1＝拡張RAM

Maj　：長辺ドット数　(0〜1023)
Min　：短辺ドット数　(0〜511)

MAJ：0＝長辺はX軸と平行，1＝長辺はY軸と平行／または長辺＝短辺

DIX　：基準点からの終点の方向　　0＝右，1＝左
DIY　：基準点からの終点の方向　　0＝下，1＝上

DX　：基準点　X座標　(0〜511)
DY　：基準点　Y座標　(0〜1023)

図4.94　LINEコマンドの動作

▶LINEレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX 基準点
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY 基準点
R#39	0	0	0	0	0	0	DY9	DY8	
R#40	MJ7	MJ6	MJ5	MJ4	MJ3	MJ2	MJ1	MJ0	Maj (NX) →長辺ドット数
R#41	0	0	0	0	0	0	MJ9	MJ8	
R#42	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0	Min (NY) →短辺ドット数
R#43	0	0	0	0	0	0	0	MI8	
R#44	0	0	0	0	C3	C2	C1	C0	CLR (GRAPHIC4, 6) 色コードデータ
	0	0	0	0	0	0	C1	C0	CLR (GRAPHIC5) 色コードデータ
	C7	C6	C5	C4	C3	C2	C1	C0	CLR (GRAPHIC7) 色コードデータ
R#45	0	—	MXD	—	DIY	DIX	—	MAJ	ARG (Argument register)

MAJ → 長辺選択
DIX → 終点方向（X）
DIY → 終点方向（Y）
MXD → メモリ選択

▶LINEコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	0	1	1	1	LO3	LO2	LO1	LO0	CMR

LO3–LO0 → ロジカルオペレーション

図4.95　LINEコマンドのレジスタ設定

リスト4.16　LINEコマンドの実行例

```
;=============================================================
;  List 4.16   LINE sample
;                 to use, set H,L,D,E,B,A and go
;                 draw LINE (H,l)-(D,E) with color B, log-OP A
;=============================================================
;
RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program satrt -----

LINE:   DI                      ;disable interrupt
        PUSH    AF              ;save LOGICAL OPRATION
        PUSH    BC              ;save COLOR
        CALL    WAIT.VDP        ;wait end of command
```

```
        LD      A,(WRVDP)
        LD      C,A
        INC     C               ;C := PORT#1's address
        LD      A,36
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A           ;R#17 := 36

        INC     C
        INC     C               ;C := PORT#3's address
        XOR     A
        OUT     (C),H           ;DX
        OUT     (C),A
        OUT     (C),L           ;DY
        OUT     (C),A

        LD      A,H             ;make DX and DIX
        SUB     A,D
        LD      D,00000100B
        JP      NC,LINE1
        LD      D,00000000B
        NEG
LINE1:  LD      H,A             ;H := DX , D := DIX

        LD      A,L             ;make DY and DIY
        SUB     A,E
        LD      E,00001000B
        JP      NC,LINE2
        LD      E,00000000B
        NEG
LINE2:  LD      L,A             ;L := DY , E := DIY

        CP      H               ;make Maj and Min
        JP      C,LINE3
        XOR     A
        OUT     (C),L           ;long side
        OUT     (C),A
        OUT     (C),H           ;short side
        OUT     (C),A
        LD      A,00000001B     ;MAJ := 1
        JP      LINE4

LINE3:  XOR     A
        OUT     (C),H
        OUT     (C),A
        OUT     (C),L
        OUT     (C),A
        LD      A,00000000B     ;MAJ := 0

LINE4:  OR      D
        OR      E               ;A := DIX , DIY , MAJ
        POP     HL              ;H := COLOR
        OUT     (C),H
        OUT     (C),A
        POP     AF              ;A := LOGICAL OPERATION
        OR      01110000B
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
```

```
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

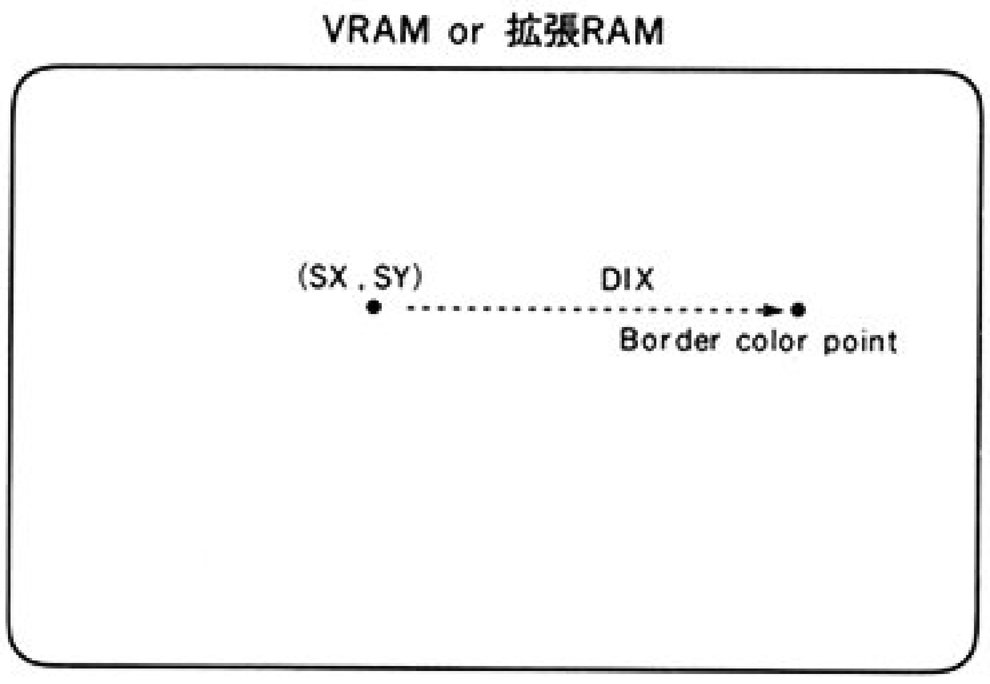
        END
```

6.5.10 SRCH(色コードのサーチ)

VRAM上の任意の座標から右または左に向かって，指定した色または指定した以外の色が存在するか否かを調べます(図4.96)．ペイントルーチンなどに使用するとたいへんに便利なコマンドです．

図4.97に示すパラメータを設定した後，R#46に60Hを書き込むとコマンドを実行します．コマンドは目的の色が見つかるか，あるいは画面の端までサーチしても見つからなかった場合終了します．S#2のCEビットが"1"の間はコマンド実行中であることを示します(図4.98)．

コマンド終了後，S#2のBDビットが"1"であれば目的の色コードが発見できたことを示し，その座標がS#8とS#9に格納されています．リスト4.17にSRCHの例を示します．



MXD：サーチ　メモリ選択　　0＝VRAM，1＝拡張RAM

SX　：サーチ基準点　X座標　(0〜511)
SY　：サーチ基準点　Y座標　(0〜1023)

DIX　：基準点からのサーチ方向　　0＝右，1＝左

EQ　：0＝境界色を発見したときに実行を終了する．
　　　1＝境界色以外を発見したときに実行を終了する．

CLR(R#44：Color register)：境界色

図4.96　SRCHコマンドの動作

▶SRCHレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB		
R#32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	SX	サーチ基準点
R#33	0	0	0	0	0	0	0	SX8		
R#34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY	
R#35	0	0	0	0	0	0	SY9	SY8		

R♯44	0	0	0	0	C3	C2	C1	C0	CLR (GRAPHIC4, 6) 境界色
	0	0	0	0	0	0	C1	C0	CLR (GRAPHIC5) 境界色
	C7	C6	C5	C4	C3	C2	C1	C0	CLR (GRAPHIC7) 境界色
R♯45	−	−	MXD	−	−	DIX	EQ	−	ARG (Argument register)

- EQ → 実行終了条件
- DIX → サーチ方向（X）
- MXD → サーチメモリ選択

▶SRCHコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R♯46	0	1	1	0	−	−	−	−	CMR
S♯2	−	−	−	BD	−	−	−	CE	
S♯8	BX7	BX6	BX5	BX4	BX3	BX2	BX1	BX0	境界色発見X座標
S♯9	1	1	1	1	1	1	BX9	BX8	境界色発見X座標

- CE → コマンド終了時：0
- BD → 境界色発見時　：1

図4.97　SRCHコマンドのレジスタ設定

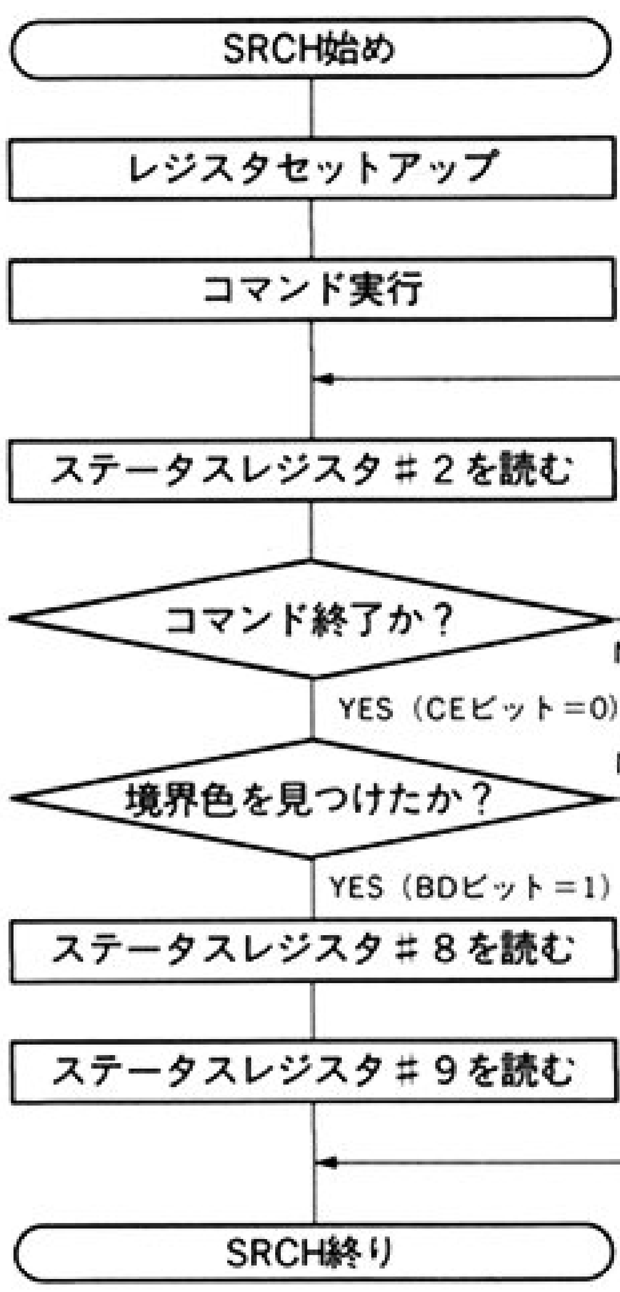


図4.98　SRCHコマンド実行のフローチャート

リスト4.17 SRCHコマンドの実行例

```
;=================================================
;  List 4.17    SRCH sample
;                to use, set H,L,E,A as follows
;                srch(x:H,y:L,color:E,arg(reg#45):A)
;                returns: Z  (not found)
;                         NZ (A := X)
;=================================================
;
        PUBLIC  SRCH

RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start -----

SRCH:   DI                      ;disable interrupt
        PUSH    AF              ;save arg
        CALL    WAIT.VDP

        LD      A,(WRVDP)
        LD      C,A
        INC     C               ;C := PORT#1's address

        LD      D,0
        LD      A,32+80H
        OUT     (C),H
        OUT     (C),A           ;R#32 := H
        INC     A
        OUT     (C),D
        OUT     (C),A           ;R#33 := 0
        INC     A
        OUT     (C),L
        OUT     (C),A           ;R#34 := L
        INC     A
        OUT     (C),D
        OUT     (C),A           ;R#35 := 0
        LD      A,44+80H
        OUT     (C),E
        OUT     (C),A           ;R#44 := E
        INC     A
        LD      E,A
        POP     AF              ;A := ARG
        OUT     (C),A
        OUT     (C),E           ;R#45 := A

        LD      A,01100000B
        OUT     (C),A
        INC     E
        OUT     (C),E           ;R#46 := SRCH command

LOOP:   LD      A,2
        CALL    GET.STATUS
        BIT     0,A
        JP      NZ,LOOP
        LD      E,A
        LD      A,8
        CALL    GET.STATUS
        LD      D,A
        LD      A,0
        CALL    GET.STATUS
        LD      A,D
        BIT     4,E

        EI
        RET
```

```
GET.STATUS:
          PUSH    BC
          LD      BC,(RDVDP)
          INC     C
          OUT     (C),A
          LD      A,8FH
          OUT     (C),A
          IN      A,(C)
          POP     BC
          RET

WAIT.VDP:
          LD      A,2
          CALL    GET.STATUS
          AND     1
          JP      NZ,WAIT.VDP
          XOR     A
          CALL    GET.STATUS
          RET

          END
```

リスト4.18 SRCHとLINEを利用した簡易PAINTルーチン

```
;=============================================================
;  List 4.18  SRCH and LINE sample
;                search color to right and left,
;                then draw line between the two point
;=============================================================
;
         EXTRN   SRCH
         EXTRN   LINE

Y        EQU     0A800H
X        EQU     0A801H
COL      EQU     0A802H
ARG      EQU     0A803H
PCOL     EQU     0A804H

;----- program start -----

MAIN:    LD      (STK),SP
         LD      SP,AREA
         LD      HL,(Y)
         LD      A,(COL)
         LD      E,A
         LD      A,(ARG)
         PUSH    HL
         PUSH    DE
         SET     2,A
         CALL    SRCH
         POP     DE
         POP     HL
         JP      NZ,S1
         LD      A,(X)
         DEC     A
S1:      INC     A
         PUSH    AF
         LD      A,(ARG)
```

```
        RES     2,A
        CALL    SRCH
        JP      NZ,S2
        LD      A,(X)
        INC     A
S2:     DEC     A
        LD      D,A
        POP     AF
        LD      H,A
        LD      A,(Y)
        LD      L,A
        LD      E,A
        LD      A,(PCOL)
        LD      B,A
        LD      A,0             ;PSET
        CALL    LINE
        LD      SP,(STK)
        RET

;----- work area -----

STK:    DS      2
        DS      200
AREA:   $

        END
```

リスト4.19　簡易PAINTルーチンの使用例

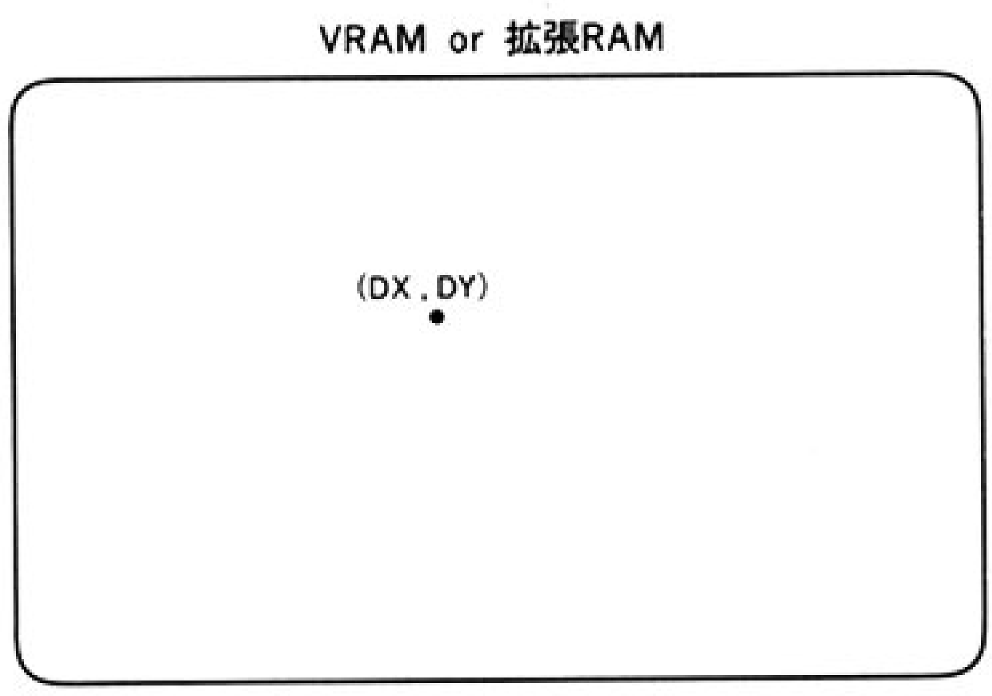
```
1000 '==============================================
1010 ' list 4.19   SRCH and LINE sample
1020 '      ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾅｶﾞﾗ ｶｰｿﾙ ｦ ｿｳｻ ｼﾃ ｸﾀﾞｻｲ
1030 '==============================================
1040 '
1050 SCREEN 5
1060 FOR I=0 TO 50:LINE -(RND(1)*255,RND(1)*211),15:NEXT
1070 I=&HA000 :DEF USR=I
1080 READ A$
1090 IF  A$="END" THEN 1130
1100   POKE I,VAL("&H"+A$):I=I+1
1110   READ A$
1120 GOTO 1090
1130 X=128:Y=100:COL=15:PCOL=2:ARG=0
1140 CURS=0
1150 A=STICK(0)
1160 CURS=(CURS+1) AND 1
1170 LINE (X-5,Y)-(X+5,Y),15,,XOR
1180 LINE (X,Y-5)-(X,Y+5),15,,XOR
1190 IF CURS=1 THEN 1290
1200 IF A=1 THEN Y=Y-1
1210 IF A=2 THEN Y=Y-1:X=X+1
1220 IF A=3 THEN X=X+1
1230 IF A=4 THEN X=X+1:Y=Y+1
1240 IF A=5 THEN Y=Y+1
1250 IF A=6 THEN Y=Y+1:X=X-1
1260 IF A=7 THEN X=X-1
1270 IF A=8 THEN X=X-1:Y=Y-1
1280 IF STRIG(0) THEN GOSUB 1300
1290 GOTO 1150
```

```
1300 POKE &HA800,Y
1310 POKE &HA801,X
1320 POKE &HA802,COL
1330 POKE &HA803,ARG
1340 POKE &HA804,PCOL
1350 A=USR(0)
1360 RETURN
1370 DATA ED,73,80,A8,31,4A,A9,2A,00,A8,3A,02
1380 DATA A8,5F,3A,03,A8,E5,D5,CB,D7,CD,AD
1390 DATA A0,D1,E1,C2,21,A0,3A,01,A8
1400 DATA 3D,3C,F5,3A,03,A8,CB,97,CD,AD,A0,C2
1410 DATA 32,A0,3A,01,A8,3C,3D,57,F1,67,3A
1420 DATA 00,A8,6F,5F,3A,04,A8,47,3E
1430 DATA 00,CD,49,A0,ED,7B,80,A8,C9,F3,F5,CD
1440 DATA 0D,A1,C5,3A,06,00,4F,0C,3E,24,ED
1450 DATA 79,3E,91,ED,79,0C,0C,AF,ED
1460 DATA 61,ED,79,ED,69,ED,79,7C,92,16,04,D2
1470 DATA 72,A0,16,00,ED,44,67,7D,93,1E,08
1480 DATA D2,7E,A0,1E,00,ED,44,BC,DA
1490 DATA 90,A0,ED,79,AF,ED,79,ED,61,ED,79,26
1500 DATA 01,C3,9C,A0,ED,61,67,AF,ED,79,ED
1510 DATA 61,ED,79,26,00,7C,B2,B3,E1
1520 DATA ED,61,ED,79,F1,E6,0F,F6,70,ED,79,FB
1530 DATA C9,F5,F3,CD,0D,A1,ED,4B,06,00,0C
1540 DATA 3E,A0,16,00,ED,61,ED,79,3C
1550 DATA ED,51,ED,79,3C,ED,69,ED,79,3C,ED,51
1560 DATA ED,79,3E,AC,ED,59,ED,79,3C,5F,F1
1570 DATA ED,79,ED,59,3E,60,ED,79,1C
1580 DATA ED,59,3E,02,CD,FD,A0,CB,47,C2,E2,A0
1590 DATA 5F,3E,08,CD,FD,A0,57,3E,00,CD,FD
1600 DATA A0,7A,CB,63,FB,C9,C5,ED,4B
1610 DATA 06,00,0C,ED,79,3E,8F,ED,79,ED,78,C1
1620 DATA C9,3E,02,CD,FD,A0,E6,01,C2,0D,A1
1630 DATA AF,CD,FD,A0,C9,END
```

6.5.11　PSET(点の描画)

VRAM上の任意の座標に点を描きます(図4.99).

図4.100に示すパラメータを設定した後, R#46に5XH(Xはロジカルオペレーション)を書き込むとコマンドを実行します. S#2のCEビットが"1"の間はコマンド実行中であることを示します. リスト4.20にPSETの例を示します.



MXD : メモリ選択　0＝VRAM, 1＝拡張RAM
DX　: 基準点　X座標　(0～511)
DY　: 基準点　Y座標　(0～1023)

図4.99　PSETコマンドの動作

▶PSETレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R#36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	DX（基準点）
R#37	0	0	0	0	0	0	0	DX8	
R#38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	DY（基準点）
R#39	0	0	0	0	0	0	DY9	DY8	
R#44	0	0	0	0	C3	C2	C1	C0	CLR (GRAPHIC4, 6)（色コードデータ）
	0	0	0	0	0	0	C1	C0	CLR (GRAPHIC5)（色コードデータ）
	C7	C6	C5	C4	C3	C2	C1	C0	CLR (GRAPHIC7)（色コードデータ）
R#45	0	−	MXD	−	−	−	−	−	ARG (Argument register)

MXD → メモリ選択

▶PSETコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R#46	0	1	0	1	LO3	LO2	LO1	LO0	CMR

LO3〜LO0 → ロジカルオペレーション

図4.100　PSETコマンドのレジスタ設定

リスト4.20　PSETコマンドの実行例

```
;================================================
;  List 4.20    PSET sample
;                  to use, set H,L,E,A as follows
;                  pset(x:H, y:L), color:E, logi-OP:A
;================================================
;
        PUBLIC  PSET

RDVDP   EQU     0006H
WRVDP   EQU     0007H

;----- program start

PSET:   DI
        PUSH    AF
        CALL    WAIT.VDP
        LD      BC,(WRVDP)

        INC     C
        LD      A,36
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A

        PUSH    BC
        INC     C
        INC     C
```

```
        XOR     A
        OUT     (C),H
        OUT     (C),A
        OUT     (C),L
        OUT     (C),A
        POP     BC

        LD      A,44
        OUT     (C),A
        LD      A,80H+17
        OUT     (C),A

        INC     C
        INC     C
        OUT     (C),E
        XOR     A
        OUT     (C),A

        LD      E,01010000B
        POP     AF
        OR      E
        OUT     (C),A

        EI
        RET

GET.STATUS:
        PUSH    BC
        LD      BC,(RDVDP)
        INC     C
        OUT     (C),A
        LD      A,8FH
        OUT     (C),A
        IN      A,(C)
        POP     BC
        RET

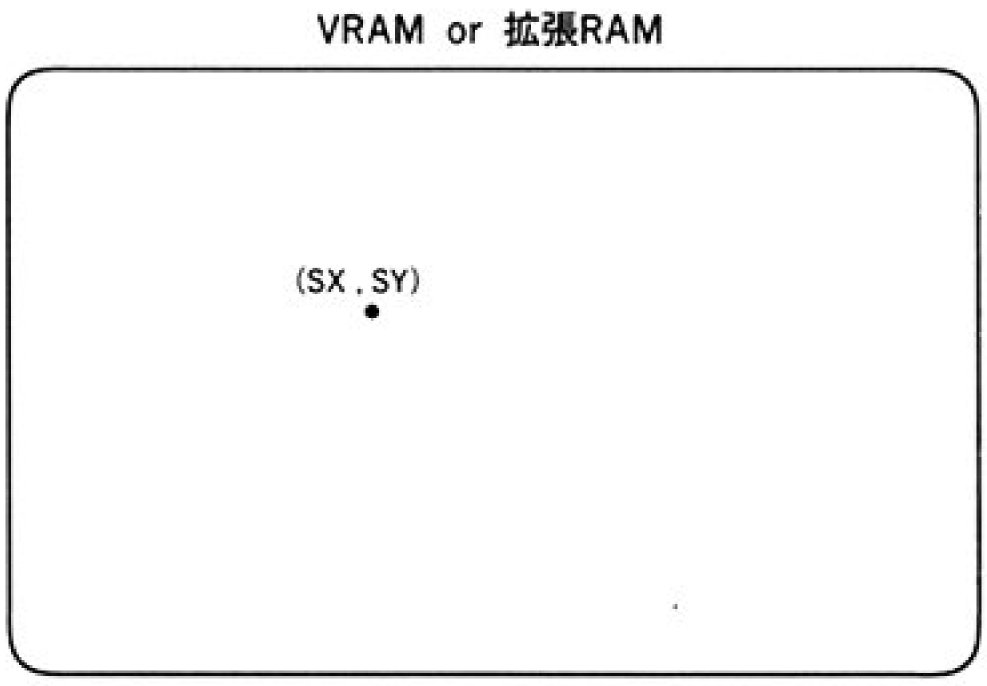
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

6.5.12 POINT(色コードの読み出し)

VRAM上の任意の座標の色コードを読み出します(**図4.101**).

図4.102に示すパラメータを設定した後，R#46に40Hを書き込むとコマンドを実行します．S#2のCEビットが"1"の間はコマンド実行中であることを示します．コマンド終了後，S#7に指定した座標の色コードが格納されています．**リスト4.21**にPOINTの例を示します．



MXS：メモリ選択　　0＝VRAM，1＝拡張RAM

SX　：基準点　X座標　(0～511)
SY　：基準点　Y座標　(0～1023)

図4.101　POINTコマンドの動作

▶POINTレジスタセットアップ

	MSB 7	6	5	4	3	2	1	0 LSB	
R♯32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	SX 基準点
R♯33	0	0	0	0	0	0	0	SX8	SX 基準点
R♯34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	SY 基準点
R♯35	0	0	0	0	0	0	SY9	SY8	SY 基準点
R♯45	－	－	－	MXS	－	－	－	－	ARG (Argument register)

MXS → メモリ選択

▶POINTコマンド実行

	MSB 7	6	5	4	3	2	1	0 LSB	
R♯46	0	1	0	0	－	－	－	－	CMR
S♯2	－	－	－	－	－	－	－	CE	
S♯7	0	0	0	0	C3	C2	C1	C0	CL (GRAPHIC4, 6) 色コードデータ
	0	0	0	0	0	0	C1	C0	CL (GRAPHIC5) 色コードデータ
	C7	C6	C5	C4	C3	C2	C1	C0	CL (GRAPHIC7) 色コードデータ

CE → コマンド終了時：0

図4.102　POINTコマンドのレジスタ設定

リスト4.21　POINTコマンドの実行例

```
;=================================================
;   List 4.21    POINT sample
;                 to use, set H,L as follows
;                 point ( x:H, y:L )
;                 returns:   A := COLOR CODE
;=================================================
;
         PUBLIC  POINT

RDVDP    EQU     0006H
WRVDP    EQU     0007H

;----- program start -----

POINT:   DI
         CALL    WAIT.VDP

         LD      A,(WRVDP)
         LD      C,A

         INC     C
         LD      A,32
         OUT     (C),A
         LD      A,80H+17
         OUT     (C),A

         INC     C
         INC     C
         XOR     A
         OUT     (C),H
         OUT     (C),A
         OUT     (C),L
         OUT     (C),A

         DEC     C
         DEC     C
         OUT     (C),A
         LD      A,80H+45
         OUT     (C),A
         LD      A,01000000B
         OUT     (C),A
         LD      A,80H+46
         OUT     (C),A
         CALL    WAIT.VDP
         LD      A,7
         CALL    GET.STATUS
         PUSH    AF
         XOR     A
         CALL    GET.STATUS
         POP     AF

         EI
         RET

GET.STATUS:
         PUSH    BC
         LD      BC,(RDVDP)
         INC     C
         OUT     (C),A
         LD      A,8FH
         OUT     (C),A
         IN      A,(C)
         POP     BC
         RET
```

```
WAIT.VDP:
        LD      A,2
        CALL    GET.STATUS
        AND     1
        JP      NZ,WAIT.VDP
        XOR     A
        CALL    GET.STATUS
        RET

        END
```

リスト4.22　PSETとPOINTを利用したPAINTルーチン

```
;==============================================================
:  List 4.22  paint routine with POINT and Pset
:       ENTRY:  X:H, Y:L, BORDER COLOR:D, PAINT COLOR:E
;==============================================================
:
        EXTRN   PSET
        EXTRN   POINT

Q.LENGTH EQU    256*2*2
MAX.Y    EQU    211


;----- paint main routine -----

PAINT:  CALL    POINT
        CP      D
        RET     Z
        CALL    INIT.Q
        LD      (COL),DE
        CALL    PUT.Q
        LD      A,(COL)
        LD      E,A
        XOR     A               ;logi-OP : PSET
        CALL    PSET
PAINT0: CALL    GET.Q
        RET     C
        INC     H
        CALL    NZ,PAINT.SUB
        DEC     H
        JP      Z,PAINT1
        DEC     H
        CALL    PAINT.SUB
        INC     H
PAINT1: DEC     L
        LD      A,-1
        CP      L
        CALL    NZ,PAINT.SUB
        INC     L
        INC     L
        LD      A,MAX.Y
        CP      L
        CALL    NC,PAINT.SUB
        JP      PAINT0
```

```
;----- check point and pset -----

PAINT.SUB:
        CALL    POINT
        LD      D,A
        LD      A,(BORD)
        CP      D
        RET     Z
        LD      A,(COL)
        CP      D
        RET     Z
        LD      E,A
        XOR     A
        CALL    PSET
        CALL    PUT.Q
        RET

;----- init Q.BUFFER pointer -----

INIT.Q:
        PUSH    HL
        LD      HL,Q.BUF
        LD      (Q.TOP),HL
        LD      (Q.BTM),HL
        POP     HL
        RET

;----- put point to Q.BUF (X:H , Y:L) ----

PUT.Q:
        EX      DE,HL
        LD      HL,(Q.TOP)
        LD      BC,Q.BUF+Q.LENGTH+1
        OR      A                       ;clear CARRY
        PUSH    HL
        SBC     HL,BC
        POP     HL
        JP      C,PUT.Q1
        LD      HL,Q.BUF
PUT.Q1:
        LD      (HL),D
        INC     HL
        LD      (HL),E
        INC     HL
        LD      (Q.TOP),HL
        EX      DE,HL
        RET

;----- take point data to D,E -----
;       returns:  NC    H:x , L:y
;                 C     buffer empty

GET.Q:  LD      HL,(Q.BTM)
        LD      BC,(Q.TOP)
        OR      A
        SBC     HL,BC
        JP      NZ,GET.Q0
        SCF
        RET

GET.Q0: LD      HL,(Q.BTM)
        LD      BC,Q.BUF+Q.LENGTH+1
        OR      A
        PUSH    HL
```

```
        SBC     HL,BC
        POP     HL
        JP      C,GET.Q1
        LD      HL,Q.BUF
GET.Q1: LD      D,(HL)
        INC     HL
        LD      E,(HL)
        INC     HL
        LD      (Q.BTM),HL
        OR      A
        EX      DE,HL
        RET

;----- work area -----

COL     DS      1
BORD    DS      1
Q.TOP   DS      2
Q.BTM   DS      2
Q.BUF   DS      Q.LENGTH

        END
```

リスト4.23　PAINTルーチンの使用例

```
1000 '==============================================
1010 ' list 4.23  paint routine with POINT and PSET
1020 '  ﾍﾟｲﾝﾄ ｶｲｼ ｲﾁ ﾆ ｶｰｿﾙ ｦ ｱﾜｾﾃ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ
1030 '==============================================
1040 '
1050 SCREEN 5
1060 FOR I=0 TO 50
1070 LINE -(RND(1)*255,RND(1)*211),15
1080 NEXT
1090 I=&HA000 :DEF USR=I
1100 READ A$
1110 IF  A$="END" THEN 1150
1120   POKE I,VAL("&H"+A$):I=I+1
1130   READ A$
1140 GOTO 1110
1150 X=128:Y=100:COL=15:PCOL=2
1160 CURS=0
1170 A=STICK(0)
1180 CURS=(CURS+1) AND 1
1190 LINE (X-5,Y)-(X+5,Y),15,,XOR
1200 LINE (X,Y-5)-(X,Y+5),15,,XOR
1210 IF CURS=1 THEN 1310
1220 IF A=1 THEN Y=Y-1
1230 IF A=2 THEN Y=Y-1:X=X+1
1240 IF A=3 THEN X=X+1
1250 IF A=4 THEN X=X+1:Y=Y+1
1260 IF A=5 THEN Y=Y+1
1270 IF A=6 THEN Y=Y+1:X=X-1
1280 IF A=7 THEN X=X-1
1290 IF A=8 THEN X=X-1:Y=Y-1
1300 IF STRIG(0) THEN GOSUB 1320
```

```
1310 GOTO 1170
1320 POKE &HA8CA,Y
1330 POKE &HA8CB,X
1340 POKE &HA8CD,COL
1350 POKE &HA8CC,PCOL
1360 A=USR(0)
1370 RETURN
1380 DATA ED,73,00,A8,31,CA,A8,2A,CA,A8,ED,5B,CC,A8,CD,67
1390 DATA A0,ED,7B,00,A8,C9,E5,21,D4,A8,22,D0,A8,22,D2,A8
1400 DATA E1,C9,EB,2A,D0,A8,01,D5,AC,B7,E5,ED,42,E1,DA,34
1410 DATA A0,21,D4,A8,72,23,73,23,22,D0,A8,EB,C9,2A,D2,A8
1420 DATA ED,4B,D0,A8,B7,ED,42,C2,4C,A0,37,C9,2A,D2,A8,01
1430 DATA D5,AC,B7,E5,ED,42,E1,DA,5D,A0,21,D4,A8,56,23,5E
1440 DATA 23,22,D2,A8,B7,EB,C9,CD,B8,A0,BA,C8,CD,16,A0,ED
1450 DATA 53,CE,A8,CD,22,A0,3A,CE,A8,5F,AF,CD,F4,A0,CD,3D
1460 DATA A0,D8,24,C4,A1,A0,25,CA,8F,A0,25,CD,A1,A0,24,2D
1470 DATA 3E,FF,BD,C4,A1,A0,2C,2C,3E,D3,BD,D4,A1,A0,C3,7E
1480 DATA A0,CD,B8,A0,57,3A,CF,A8,BA,C8,3A,CE,A8,BA,C8,5F
1490 DATA AF,CD,F4,A0,CD,22,A0,C9,F3,CD,3A,A1,ED,4B,06,00
1500 DATA 0C,3E,20,ED,79,3E,91,ED,79,0C,0C,AF,ED,61,ED,79
1510 DATA ED,69,ED,79,0D,0D,ED,79,3E,AD,ED,79,3E,40,ED,79
1520 DATA 3E,AE,ED,79,CD,3A,A1,3E,07,CD,2A,A1,F5,AF,CD,2A
1530 DATA A1,F1,FB,C9,F3,F5,CD,3A,A1,ED,4B,06,00,0C,3E,24
1540 DATA ED,79,3E,91,ED,79,C5,0C,0C,AF,ED,61,ED,79,ED,69
1550 DATA ED,79,C1,3E,2C,ED,79,3E,91,ED,79,0C,0C,ED,59,AF
1560 DATA ED,79,1E,50,F1,B3,ED,79,FB,C9,C5,ED,4B,06,00,0C
1570 DATA ED,79,3E,8F,ED,79,ED,78,C1,C9,3E,02,CD,2A,A1,E6
1580 DATA 01,C2,3A,A1,AF,CD,2A,A1,C9
1590 DATA END
```

6.6 コマンドの高速化

MSX-VIDEOは与えられたコマンドの実行のほかに目に見えない各種の処理を行っており，これらの作業のために多少コマンドの実行速度が遅くなっています．したがってこれらの処理を止めることにより，コマンドの実行速度を速めることができます．その方法を以下に示します．

1. **スプライト表示の禁止**
   この方法は画面を表示したまま速度を上げられるので有用です．具体的には，R#8のビット1を“1”にします．

2. **画面表示の禁止**
   この方法は画面が消えてしまうので，画面の初期設定時以外には，あまり使えません．この方法を実行するには，R#1のビット6を“1”にします．

6.7 コマンド終了時のレジスタの状態

各コマンドごとのコマンド終了時におけるレジスタの状態を**表4.7**に示します．

SY*，DY*およびNYBの値は，Y方向実行ドット数をNとすると次のように計算されます．

SY*＝SY＋N，DY*＝DY＋N……DIYビットが0のとき

SY*＝SY－N，DY*＝DY－N……DIYビットが1のとき

NYB＝NY－N

注　LINEでMAJビットが0のときはN＝N－1になります．

コマンド名	SX	SY	DX	DY	NX	NY	CLR	CMR H	CLR L	ARG
HMMC	—	—	—	*	—	#	—	0	—	—
YMMM	—	*	—	*	—	#	—	0	—	—
HMMM	—	*	—	*	—	#	—	0	—	—
HMMV	—	—	—	*	—	#	—	0	—	—
LMMC	—	—	—	*	—	#	—	0	—	—
LMCM	—	*	—	—	—	#	*	0	—	—
LMMM	—	*	—	*	—	#	—	0	—	—
LMMV	—	—	—	*	—	#	—	0	—	—
LINE	—	—	—	*	—	—	—	0	—	—
SRCH	—	—	—	—	—	—	—	0	—	—
PSET	—	—	—	—	—	—	—	0	—	—
PINT	—	—	—	—	—	—	*	0	—	—

－：変化なし
：コマンド終了時の座標(SY，DY*)およびカラーコード．
#：画面端を検出したときはそのときのカウント数(NYB)となる．

表4.7　コマンド終了時のレジスタの状態

第5部

BIOSによる周辺装置のアクセス

MSXの基本的な考え方は，何度も述べたように，BIOSを通して周辺装置をアクセスすることにより，機種やバージョンの差異を吸収して互換性を高めようというものです．したがって，何をするにもまずBIOSの使用法を知っておかなければなりません．この第5部では，MSXの周辺装置をBIOSによりアクセスする方法と，そのために必要となる各周辺装置の構造を説明します．

1章 PSGと音声出力

MSXには次に示すような3系統の音声出力機能が存在します．ただし，③はMSXに標準的に付属するものではないため，本書では扱いません．ここではそれ以外の①と②の機能について説明します．

① PSGによる音声出力（3チャンネル，8オクターブ）
② 1ビットI／Oポートによる音声出力
③ MSX-AUDIOによる音声出力（FM音源）…………本書では説明しない．

1.1 PSGの機能

MSXの音楽演奏機能およびBEEP音の発生には，AY-3-8910相当のLSIが使われています．このLSIはPSG（Programable Sound Genelator）とも呼ばれ，その名が示すとおり，プログラムにより複雑な音楽やさまざまな効果音を発生することができます．その特徴をまとめると次のようになります．

- 3チャンネルのトーン発生器を持ち，各チャンネルには独立に，4096種の音階（8オクターブに相当）と16段階の音量を指定することができる．
- エンベロープパターンによって，ピアノやオルガンのような音色を出すことができる．ただし，エンベロープ発生器がひとつしか存在しないため，音色を変えられるのは基本的に1チャンネルだけである．
- 内蔵するノイズ発生器によって，風の音や波の音のような効果音も容易に得られる．ただし，ノイズ発生器がひとつしか存在しないため，ノイズを出力できるのは基本的に1チャンネルだけである．
- 入力クロックfcを分周することで，トーンやエンベロープをはじめとする必要なすべての周波数を得ている（なおMSXでは，fc=1.7897725MHzと定められている）．このため音程やリズムのふらつきがまったくない．

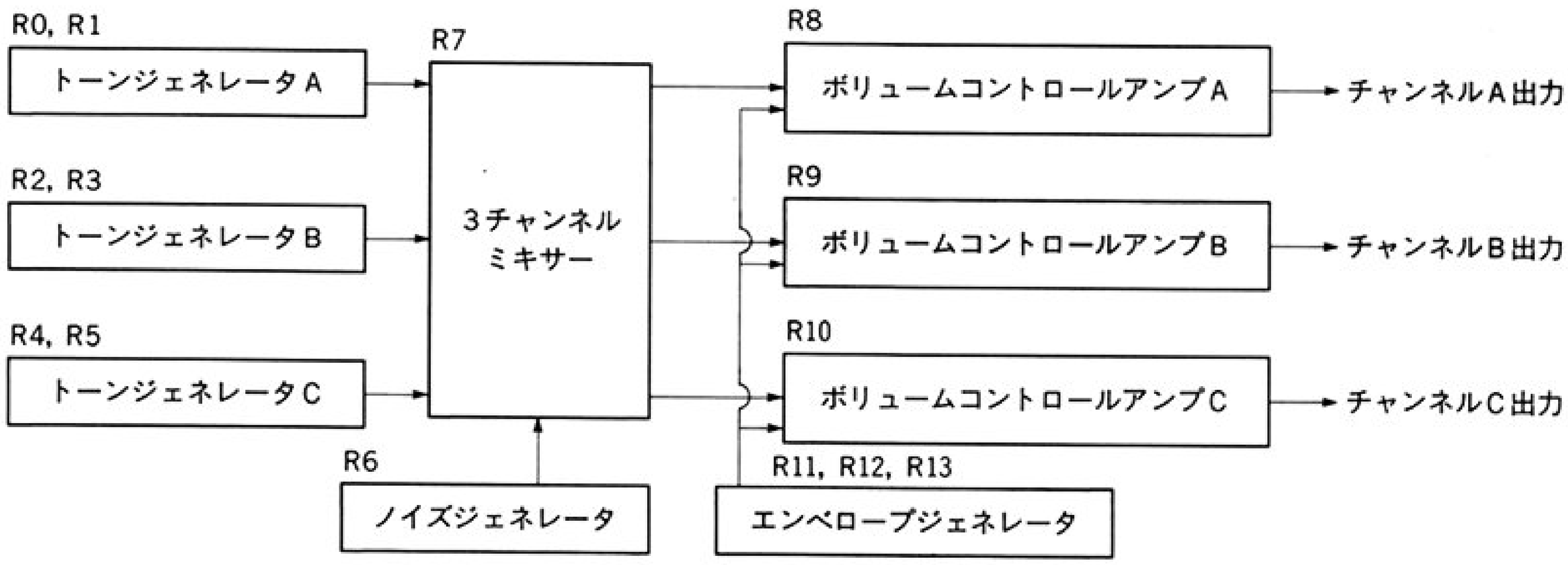


図5.1　PSGのブロックダイアグラム

上のブロックダイアグラムでは省略しましたが，PSGは音声発生の機能とはまったく別に2つの入出力ポートを持っています．MSXではこれを汎用入出力ポートとして，ジョイスティック，タッチパッド，パドル，マウスなどの入出力装置との接続にも利用しています．なお，これらの汎用入出力ポートについては，5章で説明することにします．

● PSGのレジスタ

PSGが音声出力の作業をすべて実行してくれるので，CPUの仕事は単に音が変化する時点でPSGにそれを教えることだけとなりました．これは**図5.2**に示したようなPSG内部の16個の8ビットレジスタに値を書き込むことによって行います．

以下に，これらのレジスタの役割とその使用法を説明します．

● トーン周波数の設定（R0～R5）

A，B，C各チャンネルのトーン周波数はR0～R5によって設定します．まずPSG内で入力クロック（fc=1.7897725MHz）を16分周し，それを基準周波数とします．また各チャンネルは，この基準周波数をそれぞれに割り当てられた12ビットのデータで分周して最終的な音程を得ます．12ビットの分周データ（TP）と発生する音の周波数（ft）には次のような関係があります．

```
ft=fc/(16*TP)
  =0.11186078125/TP [MHz]
  =111860.78125/TP [Hz]
```

12ビットの分周データTPは，各チャンネルごとに上位4ビットの粗調整値CTと下位8ビットの微調整値FTによって，**図5.3**のように指定されます．また，音階を作るためのレジスタの設定を**表5.1**に示します．

レジスタ ＼ ビット		B7 (MSB)	B6	B5	B4	B3	B2	B1	B0 (LSB)
R0	チャンネルA音程分周比	下位8ビット							
R1						上位4ビット			
R2	チャンネルB音程分周比	下位8ビット							
R3						上位4ビット			
R4	チャンネルC音程分周比	下位8ビット							
R5						上位4ビット			
R6	ノイズ分周比								
R7	$\overline{\text{Enable}}$	$\overline{\text{IN}}$／OUT		$\overline{\text{NOISE}}$			$\overline{\text{TONE}}$		
		IOB	IOA	C	B	A	C	B	A
R8	チャンネルA音量				M				
R9	チャンネルB音量				M				
R10	チャンネルC音量				M				
R11	エンベロープ周期	下位8ビット							
R12		上位8ビット							
R13	エンベロープ波形								
R14	I／Oポート A								
R15	I／Oポート B								

図5.2　PSGのレジスタ構成

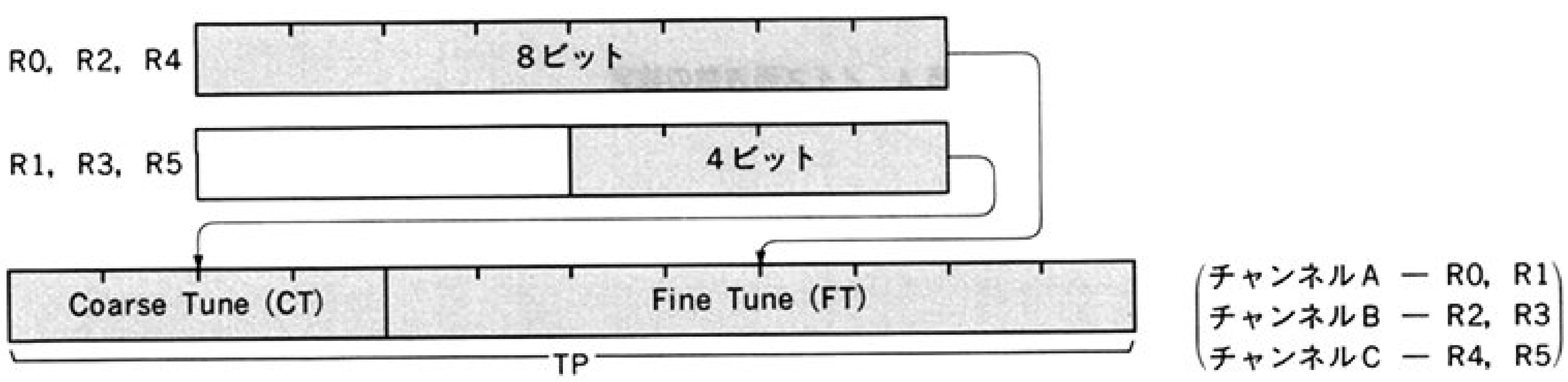


図5.3　音程の設定

音名＼オクターブ	1	2	3	4	5	6	7	8
C	D5D	6AF	357	1AC	D6	6B	35	1B
C#	C9C	64E	327	194	CA	65	32	19
D	BE7	5F4	2FA	17D	BE	5F	30	18
D#	B3C	59E	2CF	168	B4	5A	2D	16
E	A9B	54E	2A7	153	AA	55	2A	15
F	A02	501	281	140	A0	50	28	14
F#	973	4BA	25D	12E	97	4C	26	13
G	8EB	476	23B	11D	8F	47	24	12
G#	86B	436	21B	10D	87	43	22	11
A	7F2	3F9	1FD	FE	7F	40	20	10
A#	780	3C0	1E0	F0	78	3C	1E	F
B	714	38A	1C5	E3	71	39	1C	E

表5.1　トーン周波数の設定(音階データ)

● ノイズ周波数の設定（R6）

爆発音や波の音などの合成にかかせないのがノイズ音です．PSGは，ノイズジェネレータによって発生させたノイズをA～Cの各チャンネルに出力できます．ノイズジェネレータは１つしか存在しないため，各チャンネル共通に同一のノイズが出力されます．ノイズは平均周波数を変えることによって様々な効果を出すことが可能であり，R6レジスタがその設定を行います．このレジスタの下位５ビットのデータ（NP）で基準周波数（fc/16）を分周し，ノイズの平均周波数（fn）を決定します．

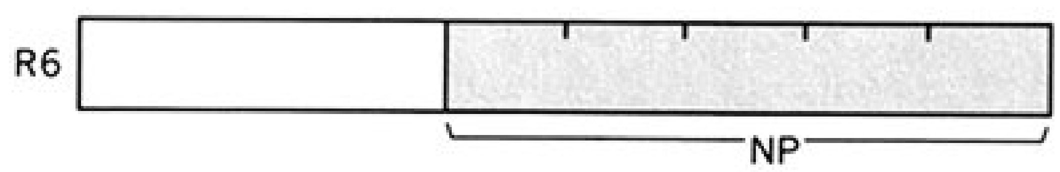


図5.4　ノイズ周波数の設定

NPとfnの間には次のような関係があります．

```
fn=fc/(16*NP)
  =0.11186078125/NP [MHz]
  =111860.78125/NP [Hz]
```

NPは1～31の値をとりますから，ノイズの平均周波数は3.6kHz～111.9kHzの範囲で設定できます．

● 音のミキシング（R7）

R7は各チャンネルごとに，トーンジェネレータとノイズジェネレータの出力を選択するためのレジスタで，ノイズとトーンの両者を混合することもできます．図5.5に示したように，R7の下位3ビット（B0〜B2）はトーン出力，次の3ビット（B3〜B5）はノイズ出力の制御を行います．どちらも対応するビットが0の時に出力ON，1の時にOFFとなります．

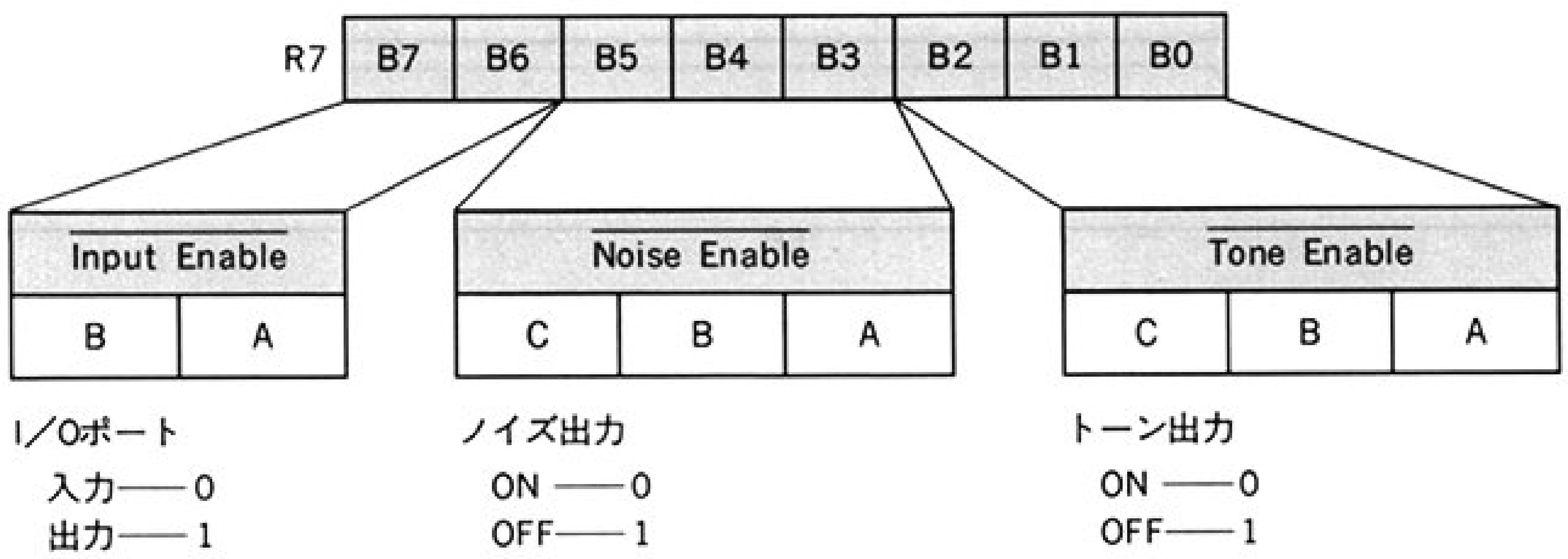


図5.5　各チャンネルの出力選択

R7の上位2ビットは音声出力とは関係ありません．これはPSGが持つ2本のI/Oポートのデータ方向を決定するもので，対応するビットが0の時に入力モード，1の時に出力モードが設定されます．MSXではポートAを入力，ポートBを出力として使用していますから，つねにビット6＝“0”，ビット7＝“1”と設定しておく必要があります．

● 音量の設定（R8〜R10）

R8〜R10は各チャンネルの音量を指定するレジスタです．4ビットのデータ（0〜15）で音量を固定的に指定する方法とエンベロープを用いてビブラートや減衰音などの効果音を発生させる方法の2通りが選べますが，その選択もこのレジスタで行います．

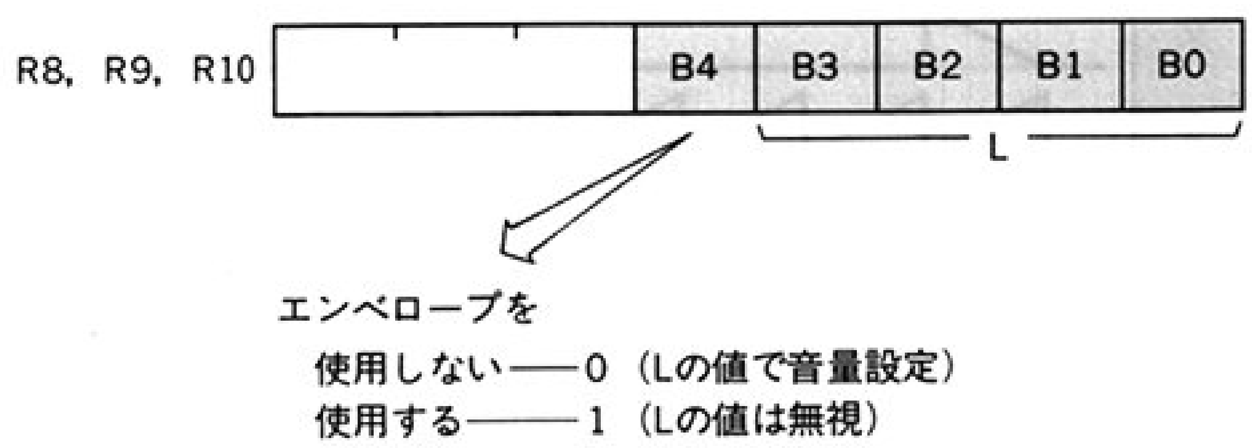


図5.6　音量の設定

なお，これらのレジスタのビット4が“0”の場合，エンベロープは使用されず，レジスタの下位4ビットの値L（0〜15）によって音量が指定されます．また，ビット4が“1”の場合はエンベロープ信号によって音量が変化し，Lの値は無視されます．

● エンベロープ周期の設定（R11，R12）

R11, R12 はエンベロープの周期を16ビットデータによって指定します．データの上位8ビットをR12で，下位8ビットをR11で設定します．

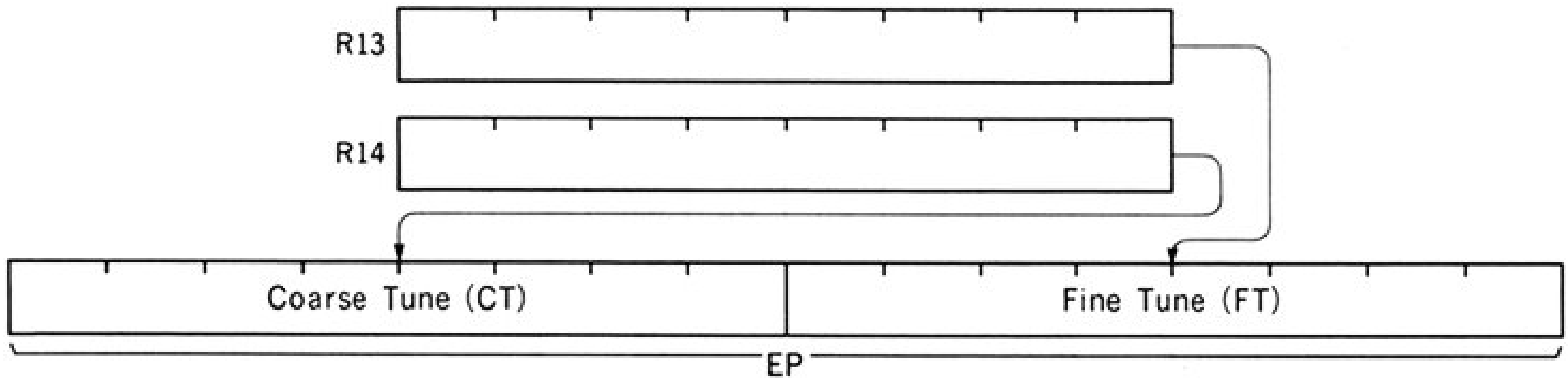


図5.7　エンベロープ周波数の設定

なお，エンベロープの周期Tと16ビットデータEPの間には，次のような関係があります．

T = (256 * EP)/fc
　= (256 * EP)/1.787725 [MHz]
　=143.03493 * EP [μs]

● エンベロープパターンの設定（R13）

R13は，下位4ビットのデータによって**図5.8**のようにエンベロープパターンを設定します．同図に示したTの間隔がR11とR12で指定されたエンベロープ周期に相当します．

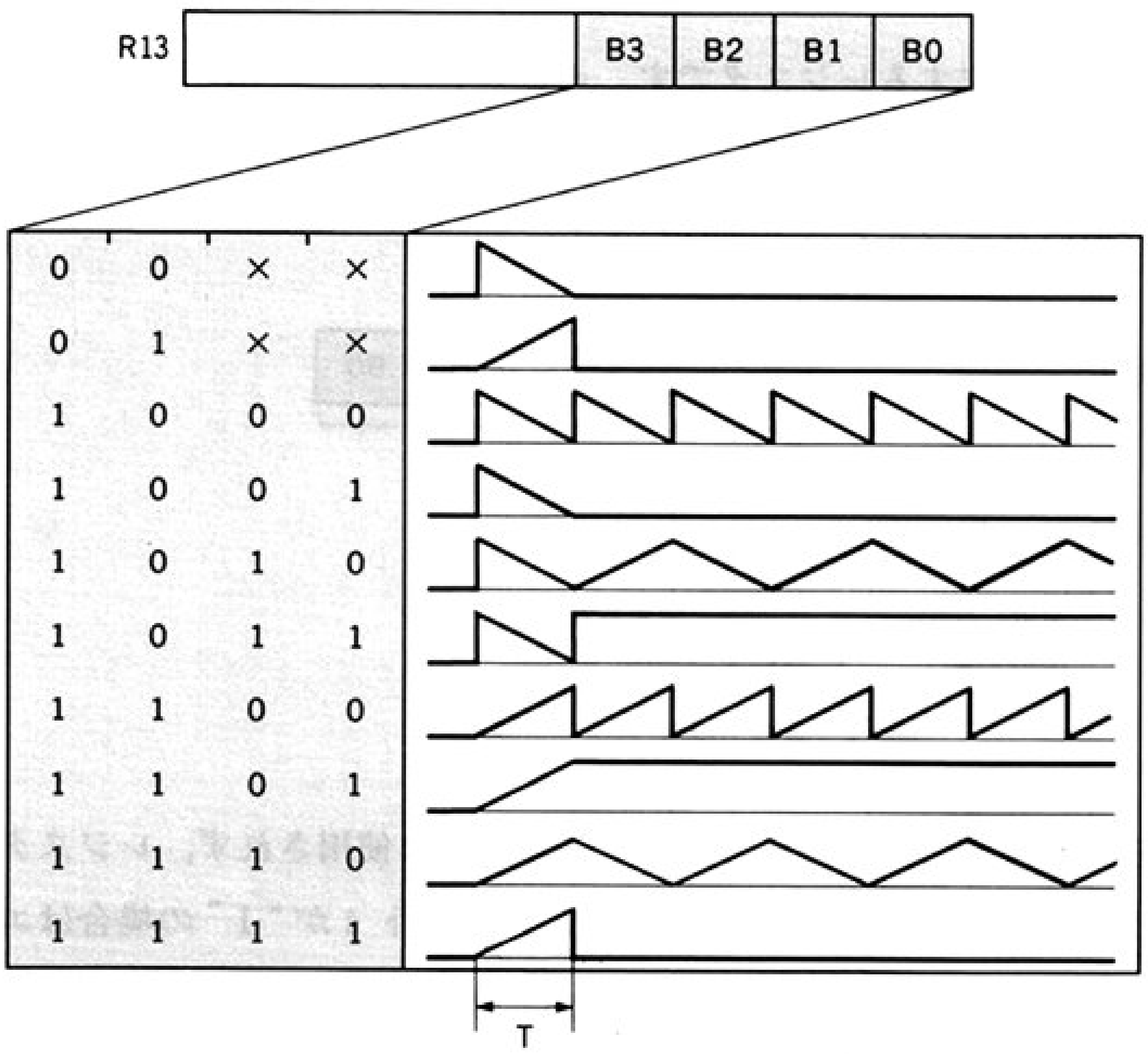


図5.8　エンベロープの波形の設定

● I／O ポート（R14，R15）

R14 と R15 は 8 ビットのデータをパラレルで入出力するポートです．MSX ではここを汎用入出力インターフェイスとして使用しています．詳しくは 5 章を参照してください．

1.2　PSG のアクセス

マシン語プログラム中から PSG をアクセスするために，以下に述べるようないくつかの BIOS ルーチンが用意されています．

● GICINI（0090H／MAIN）……………PSG の初期化

入力：——

出力：——

機能：PSG のレジスタを初期化し，さらに BASIC の PLAY 文を実行するための作業領域の初期設定を行う．この時 PSG の各レジスタは図 5.9 のような値に設定される．

レジスタ ＼ ビット		7	6	5	4	3	2	1	0
R0	チャンネルA周波数	0	1	0	1	0	1	0	1
R1		0	0	0	0	0	0	0	0
R2	チャンネルB周波数	0	0	0	0	0	0	0	0
R3		0	0	0	0	0	0	0	0
R4	チャンネルC周波数	0	0	0	0	0	0	0	0
R5		0	0	0	0	0	0	0	0
R6	ノイズ周波数	0	0	0	0	0	0	0	0
R7	チャンネル設定	1	0	1	1	1	0	0	0
R8	チャンネルA音量	0	0	0	0	0	0	0	0
R9	チャンネルB音量	0	0	0	0	0	0	0	0
R10	チャンネルC音量	0	0	0	0	0	0	0	0
R11	エンベロープ周期	0	0	0	0	1	0	1	1
R12		0	0	0	0	0	0	0	0
R13	エンベロープパターン	0	0	0	0	0	0	0	0
R14	I／Oポート A								
R15	I／Oポート B								

図5.9　PSGレジスタの初期値

● WRTPSG（0093H/MAIN）……………PSG レジスタへのデータの書き込み

入力：A←PSG のレジスタ番号

E←書き込むデータ

出力：――

機能：Aレジスタで指定した番号のPSGレジスタに，Eレジスタの内容を書き込む．

● RDPSG（0096H／MAIN）……………PSGのレジスタのデータの読み出し

入力：A←PSGのレジスタ番号

出力：A←指定したレジスタの内容

機能：Aレジスタで指定した番号のPSGレジスタの内容を読み出し，その値をAレジスタに格納する．

● STRTMS（0099H／MAIN）……………音楽演奏の開始

入力：（QUEUE）←中間言語に変換されたMML

出力：――

機能：バックグラウンドタスクとして，音楽が流れているかどうかを判定し，もし流れていなければキューに設定された音楽を演奏する．

リスト5.1　単音の発生

```
;=============================================================
;
;   List 5.1    440 Hz tone
;
;=============================================================
;
WRTPSG  EQU     0093H

        ORG     0B000H

;-------- program start --------

        LD      A,7             ;Select Channel
        LD      E,00111110B     ;Channel A Tone := On
        CALL    WRTPSG

        LD      A,8             ;Set Volume
        LD      E,10
        CALL    WRTPSG

        LD      A,0             ;Set Fine Tune Channel A
        LD      E,0FEH          ;Data 0FEH
        CALL    WRTPSG

        LD      A,1             ;Set Coarse Tune Channel A
        LD      E,0             ;Data 0H
        CALL    WRTPSG

        RET

        END
```

1.3　1ビットサウンドポートによる音声発生機能

MSXには，PSGのほかにもうひとつ音源が存在しています．これは1ビットのI／Oポートの出力をソフトで繰り返しON／OFFして音を出すという単純なものです．

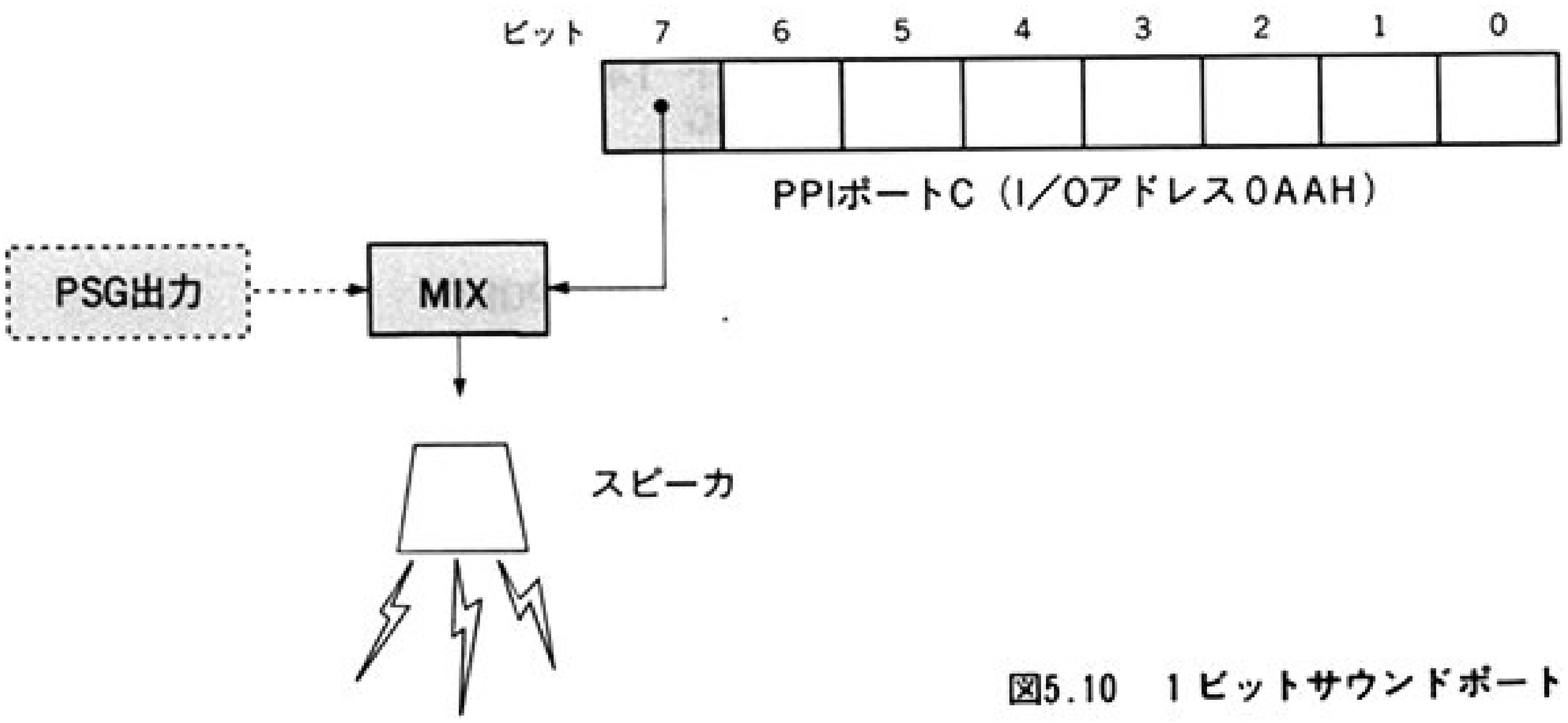


図5.10　1ビットサウンドポート

1.4　1ビットサウンドポートのアクセス

この1ビットサウンドポートをアクセスするために，次に示すBIOSルーチンが用意されています．

● CHGSND（0135H／MAIN）

入力：A←ON／OFF指定（0＝OFF，0以外＝ON）

出力：——

機能：Aレジスタに0を入れてこのルーチンをコールするとサウンドポートのビットをOFFにし，0以外の値を入れてコールするとサウンドポートのビットをONにする．

リスト5.2　カセットテープの再生

```
;==========================================================
;
;  List 5.2   Read from cassette tape
;
;             Set music tape into tape-recorder
;             and run this program.
;             Then your MSX will replay it.
;
;==========================================================
;
CHGSNG  EQU     0135H
STMOTR  EQU     00F3H
```

```
RDPSG   EQU     0096H
BREAKX  EQU     00B7H

        ORG     0B000H

;----- program start -----      1ビットサウンドポートを利用してテープの音声を再生する

START:  LD      A,1             ;motor on
        CALL    STMOTR

LBL01:  LD      A,14            ;register 14
        CALL    RDPSG           ;read PSG

        AND     80H             ;check CSAR

        CALL    CHGSNG          ;change SOUND PORT

        CALL    BREAKX          ;check Ctrl-STOP
        JR      NC,LBL01

        XOR     A               ;stop cassette motor
        CALL    STMOTR
        RET

        END
```

2章 カセット・インターフェイス

カセットテープレコーダは，MSX の最も安価な外部記憶装置です．このカセットテープ上の情報をマシン語中で取り扱うためには，カセット・インターフェイスの使用法を知らなくてはなりません．本章では，このために必要となる情報を説明します．

2.1 ボーレート

MSX のカセット・インターフェイスは以下の 2 種類のボーレートを使用できます（**表5.2**）．なお，BASIC のスタート時にはデフォルトで1200ボーが設定されています．

ボーレート	特　徴
1200bps	低スピード／高信頼性
2400bps	高スピード／低信頼性

表5.2　MSXのボーレート

ボーレートの指定は，SCREEN 命令の第 4 パラメータまたは CSAVE 命令の第2パラメータで行います．一度指定すると以後そのボーレートが引き続いて使用されます．

SCREEN　　,,,<ボーレート>
CSAVE　　"ファイル名",<ボーレート>
　　　　（<ボーレート>は両者とも，1で1200ボー，2で2400ボー）

2.2 1ビットの構成

入出力の基本となる 1 ビットのデータは**図 5.11** のように記録されます．パルスの幅は CPU の T-STATE をカウントして決められるため，カセット・インターフェイス作動中はすべての割り込みが禁止されています．

なお，カセットから入力されるビットデータは，汎用入出力インターフェイスのポート B（PSG

のレジスタ#15）の第７ビットを通して読み取ることが可能です．第５部１章リスト５.３のプログラム例の中で，この機能を使用していますので，参照してください．

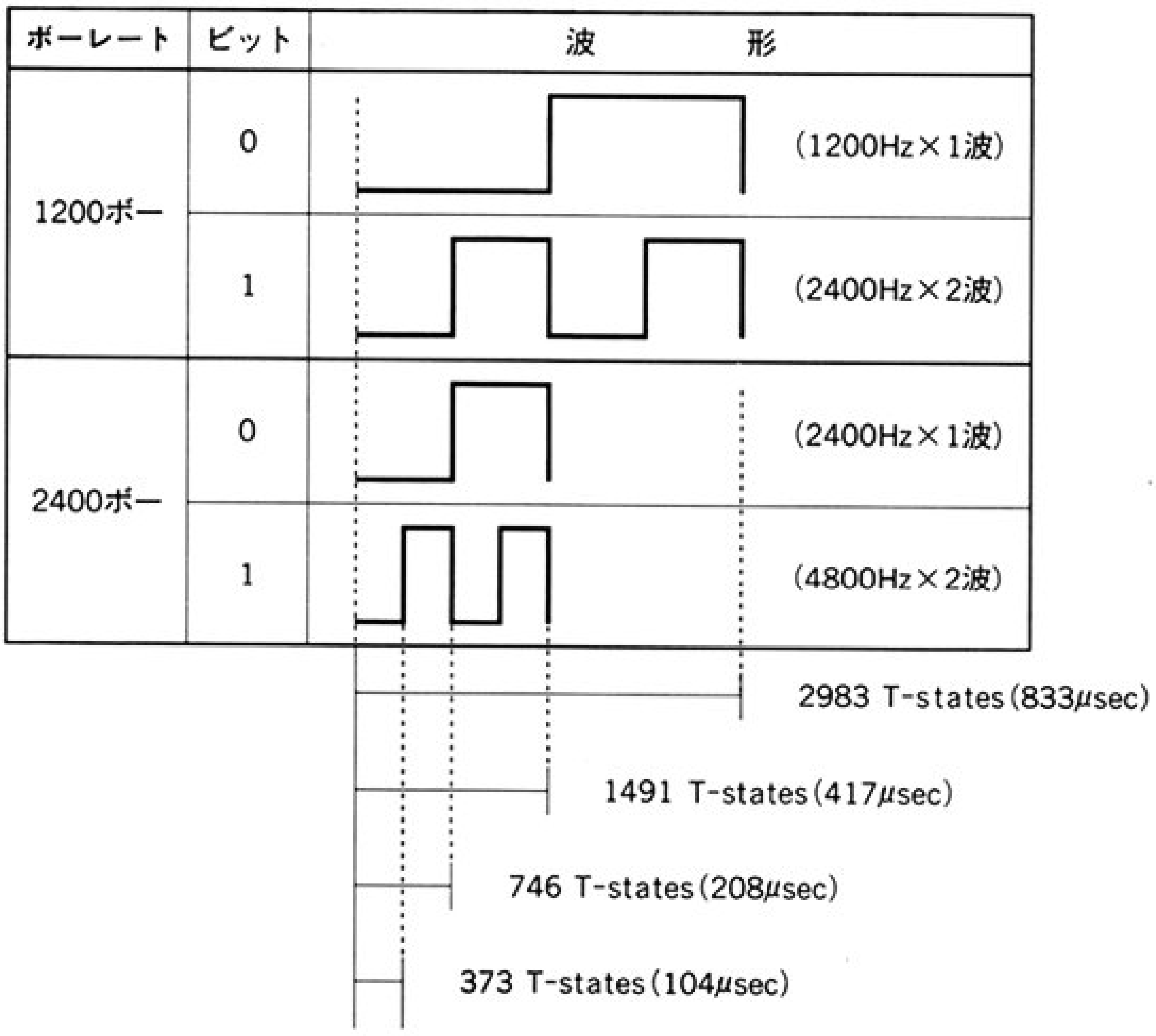


図5.11　１ビットの構成

２.３　１バイトの構成

１バイトのデータは図５.12のようなビット列で記録されます．スタートビットとして“０”のビットがひとつ，次にデータ本体が最下位ビットから最上位ビットの順で８ビット，最後にストップビットとして“１”のビットが２つ続き，合計11ビットを使用します．

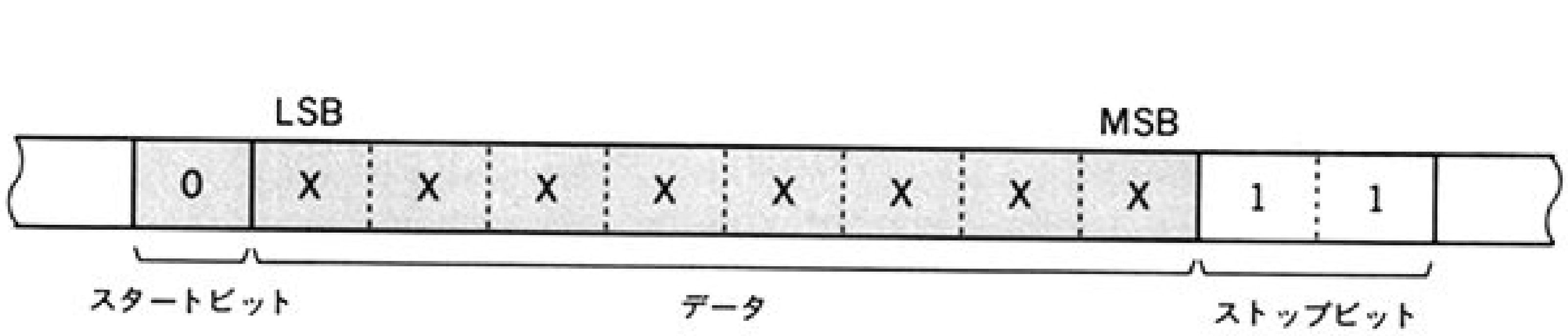


図5.12　１バイトの構成

2.4　ヘッダの構成

ヘッダとはセーブまたはロード時に，カセットテープが動き出してから回転が安定するまでの時間待ちや，ファイル間に区切りをつけるために，ある特定の周波数の信号を一定時間テープに記録させた部分のことです．ロングヘッダとショートヘッダの2種類があり，ロングヘッダはモータの回転が安定するまでの時間待ちに使用されます．また，テープリード時のボーレートは，ロングヘッダを読んで決定されます．ショートヘッダはファイルボディ間の区切りに使用されます．**表5.3**に両者の構成を示しました．

ボーレート	ヘッダの種類	ヘッダの構成
1200ボー	ロングヘッダ	2400Hz×16000波(約6.7秒)
	ショートヘッダ	2400Hz×4000波(約1.7秒)
2400ボー	ロングヘッダ	4800Hz×32000波(約6.7秒)
	ショートヘッダ	4800Hz×8000波(約1.7秒)

表5.3　ヘッダの構成

2.5　ファイルのフォーマット

MSXのBASICは次の3タイプのカセットフォーマットのファイルをサポートしています．

(1) BASICテキストファイル

CSAVE命令でバイナリセーブしたBASICプログラムは，このフォーマットにしたがって記録されます．なお，ファイルは前半のファイルヘッダと後半のファイルボディに分けられます．

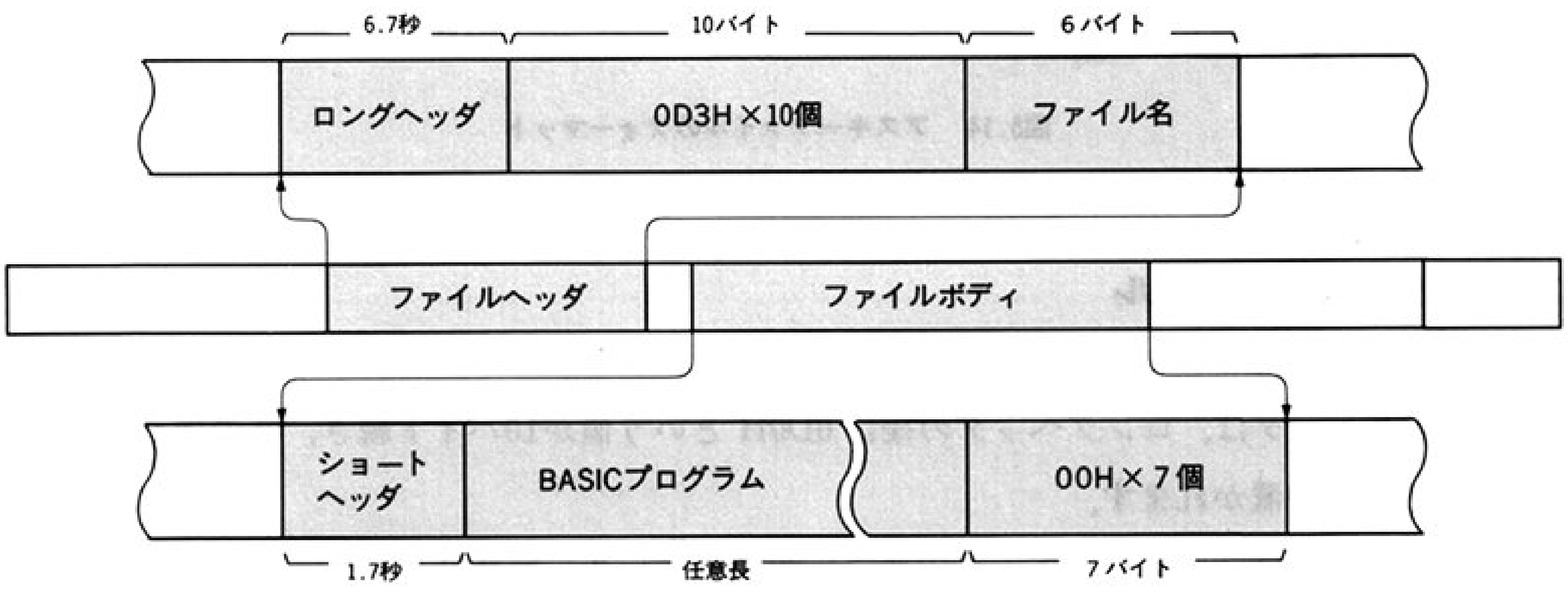


図5.13　バイナリファイルのフォーマット

ファイルヘッダは，ロングヘッダの後に0D3Hという値が10バイト続き，その後に6バイトのファイル名が置かれます．ファイルボディはショートヘッダの後にプログラム本体が続き，7バイトの00Hでファイルエンドを示します．

(2) ASCIIテキストファイル

SAVE命令でアスキーセーブされたBASICプログラム，およびOPEN命令によって作られたデータファイルは，このフォーマットにしたがって記録されます．

ファイルヘッダは，ロングヘッダの後に0EAHという値が10バイト続き，その後に6バイトのファイル名が置かれます．

データ本体は256バイトごとのブロックに分けられ，それぞれのブロックの前にはショートヘッダが置かれます．ファイルエンドはCtrl-Z (1AH) で示されます．このため，1AHという値を含むデータファイルを作ることはできません．

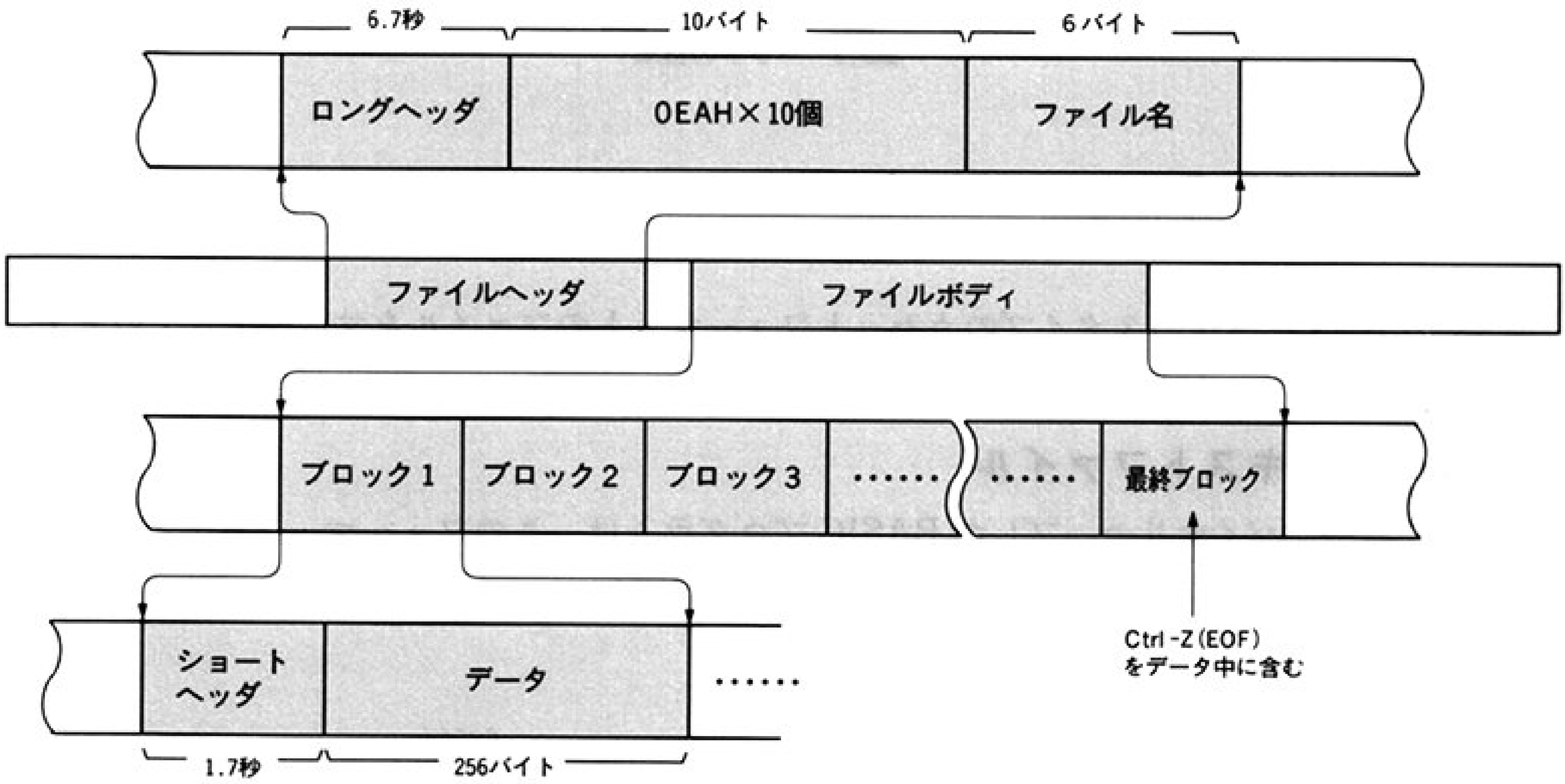


図5.14　アスキーファイルのフォーマット

(3) マシン語ファイル

BSAVE命令でセーブされたマシン語ファイルは，このフォーマットにしたがって記録されます．ファイルヘッダは，ロングヘッダの後に0D0Hという値が10バイト続き，その後ろに6バイトのファイル名が置かれます．

ファイルボディは，ショートヘッダの後に，先頭アドレス，最終アドレス，実行開始アドレスの順に3つのアドレスが記録され，その次にマシン語本体が続きます．データの量は先頭アドレ

スと最終アドレスから計算できますから，特別なファイルエンドマークはありません．実行開始アドレスとは，BLOAD命令のRオプション使用時に，プログラムが実行されるアドレスです．

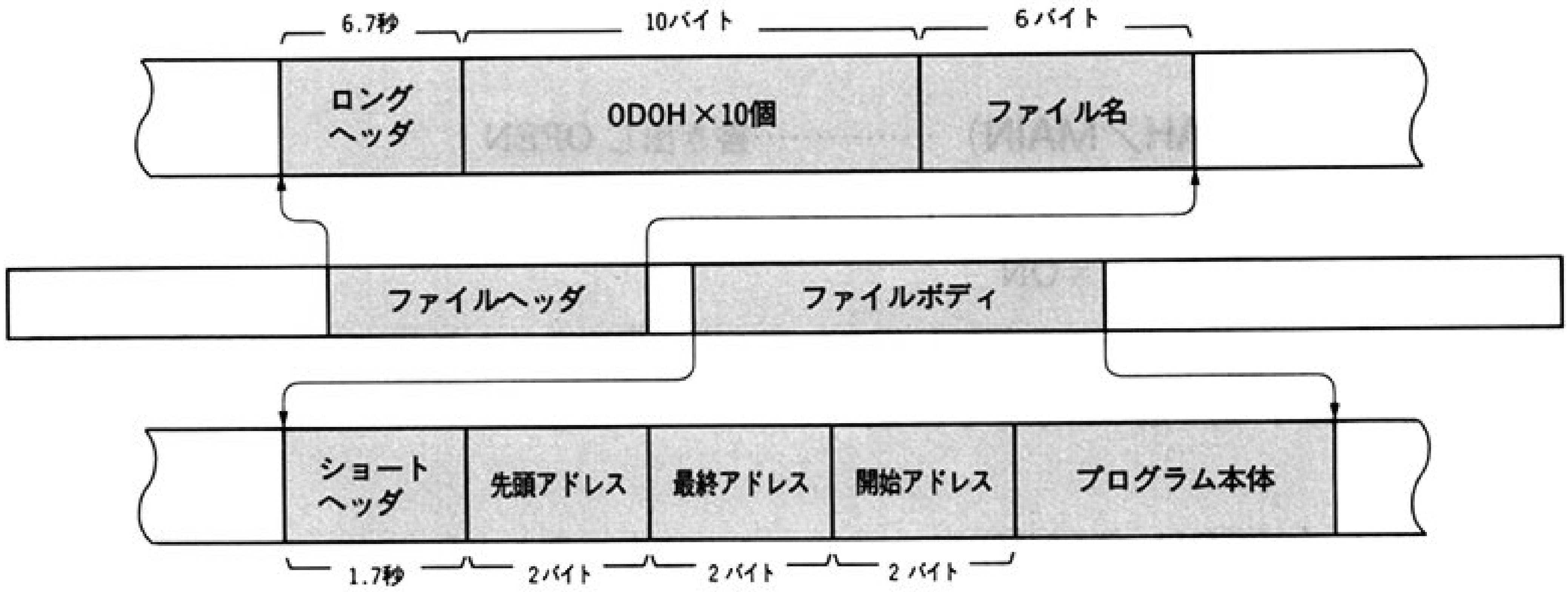


図5.15　マシン語ファイルのフォーマット

2.6　カセットファイルのアクセス

カセットファイルをアクセスするために，以下のBIOSルーチンが用意されています．

● TAPION（00E1H／MAIN）……………読み込みOPEN

入力：——

出力：異常終了時CYフラグ=ON

機能：テープレコーダのモータを起動し，ロングヘッダまたはショートヘッダを読み込む．同時にそのファイルが記録されたボーレートを判別し，ワークエリアをそれに合わせて設定する．なお，割り込みは禁止される．

● TAPIN（00E4H／MAIN）……………1バイトの読み込み

入力：——

出力：A←読み込んだデータ
異常終了時CYフラグ=ON

機能：テープからデータを1バイト読み込み，Aレジスタに格納する．

● **TAPIOF（00E7H／MAIN）**……………読み込みCLOSE

入力：――

出力：――

機能：テープからの読み込み動作を終了する．同時に割り込みは再開される．

● **TAPOON（00EAH／MAIN）**……………書き出しOPEN

入力：A←ヘッダの種類（0＝ショートヘッダ，0以外＝ロングヘッダ）

出力：異常終了時CYフラグ＝ON

機能：テープレコーダのモータを起動し，Aレジスタで指定された種類のヘッダをテープに書き出す．なお，割り込みは禁止される．

● **TAPOUT（00EDH／MAIN）**……………1バイトの書き出し

入力：A←書き込みデータ

出力：異常終了時CYフラグ＝ON

機能：Aレジスタの内容をテープに書き込む．

● **TAPOOF（00F0H／MAIN）**……………書き込みCLOSE

入力：――

出力：――

機能：テープへの書き込み動作を終了する．同時に割り込みは再開される．

● **STMOTR（00F3H／MAIN）**……………モータの動作指定

入力：A←動作の指定（0＝停止，1＝起動，255＝現在と反対の状態にする）

出力：――

機能：Aレジスタで指定した値にしたがって，モータの動作状態を設定する．

これらのBIOSを使用してカセットファイルのREAD／WRITEルーチンを作成する際，なるべく無駄な動作をせずに，ただひたすらREADまたはWRITEのみに専念するようにしてください．たとえば，テープからデータを読み込みながらそのデータをCRTに表示させたりすると，READエラーを生じることがあります．

リスト5.4にBIOSルーチンを使用したサンプルプログラムを示します．

リスト5.3　カセットにセーブされたファイル名の表示

```
;==============================================================
;
;  List 5.3       Cassete files
;
;       Set cassette tape into recorder and run this program.
;       Then the all names and atributes of the programs
;       in that tape are listed out.
;
;==============================================================
;
CHPUT   EQU     00A2H
TAPION  EQU     00E1H
TAPIN   EQU     00E4H
TAPIOF  EQU     00E7H

        ORG     0C000H

;----- program start -----          カセットテープ内のプログラム名一覧を表示する

START:  CALL    TAPION              ;motor on and read header

        LD      B,16
        LD      HL,WORK             ;work area address
LBL01:  PUSH    HL
        PUSH    BC
        CALL    TAPIN               ;read a byte of data from tape
        POP     BC
        POP     HL
        JR      C,ERROR             ;set carry flag if read error
        LD      (HL),A
        INC     HL
        DJNZ    LBL01

        LD      HL,FILNAM           ;write file name
        CALL    PUTSTR
        LD      HL,WORK+10
        CALL    PUTSTR
        CALL    CRLF

        LD      A,(WORK)            ;check file attributes

        LD      HL,BINFIL
        CP      0D3H                ;check binary file
        JR      Z,LBL03

        LD      HL,ASCFIL
        CP      0EAH                ;check ascii file
        JR      Z,LBL03

        LD      HL,MACFIL
        CP      0D0H                ;check machine code file
        JR      Z,LBL03

ERROR:  LD      HL,ERRSTR

LBL03:  CALL    PUTSTR
        CALL    TAPIOF
        RET

;----- put CRLF -----

CRLF:   LD      HL,STCRLF
        CALL    PUTSTR
        RET
```

```
;----- put string -----

PUTSTR: LD      A,(HL)          ;get a character from strings
        CP      '$'             ;check end of strings
        RET     Z
        CALL    CHPUT           ;write a character to CRT
        INC     HL
        JR      PUTSTR

;----- strings data -----

FILNAM: DB      'FILE NAME :$'
ASCFIL: DB      'ASCII FILE',0DH,0AH,'$'
BINFIL: DB      'BINARY FILE',0DH,0AH,'$'
MACFIL: DB      'BSAVE FILE',0DH,0AH,'$'
ERRSTR: DB      'TAPE READ ERROR',0DH,0AH,'$'
STCRLF: DB      0DH,0AH,'$'

;----- WORK AREA -----

WORK:   DS      16,0
        DB      '$'             ;end of strings

        END
```

3章　キーボード・インターフェイス

MSX 2のキーボードはMSX 1と同じ構造をしていますが，カナの入力時にローマ字カナ変換方式が使えるようになり，機能的にはたいへん便利になりました．本章ではMSX 2のキーボード・インターフェイスについて説明しましょう．

なお，キーの配列などについては日本仕様のキーボードをもとに説明しますが，海外仕様のMSXではデータが少し異なる部分もあります．

3.1　キー配列

MSXのキーボードの配列は，英数字はJIS標準配列を採用しており，カナはJIS標準配列と五十音順配列をジャンパ線により切り換えています．ただし，このジャンパ線による設定はシステム起動時にどちらの配列を選ぶか決めているだけであり，ワークエリアKANAMDの値で任意に変更可能です．

● **KANAMD（FCADH，1）**………キー配列（0＝五十音配列，0以外＝JIS配列）

3.2　キースキャン

MSXは図5.16のようなキーマトリクスを持っています．このキーマトリクスを調べることによって，キーの押されている状況をリアルタイムに得ることが可能であり，ゲームなどの入力に使用できます．

キーマトリクスのスキャンは，次のBIOSルーチンを用いて行います．

● **SNSMAT（0141H／MAIN）**……………キーマトリクスの指定行の読み出し

入力：A←読み出すキーマトリクスの行（0～10）

出力：A←指定したキーマトリクスの行の状態（押されているキーのビットが0になる）

機能：図5.16のキーマトリクスの1行を指定し，その状態をAレジスタに返す．この時，押されているキーに対応するビットが“0”，押されていないキーに対応するビットは“1”になる．

行	7 (MSB)	6	5	4	3	2	1	0 (LSB)
0	' 7 土 ニ	& 6 金 ナ	% 5 木 ォ オ	$ 4 水 ェ エ	# 3 火 ゥ ウ	″ 2 月 ィ イ	! 1 日 ァ ア	0 万 ノ
1	+ : ♣ モ	¦ [○ 「 ロ	' @ レ	| ¥ 円 ル	~ ^ リ	= − ー ラ	) 9 千 ネ	(8 百 ヌ
2	B ┘ ト	A サ	_ ◆ ・ ン	? / ♠ 。 ヲ	> . 大 、 ワ	< , 小 ョ ヨ	¦] ● 」 ゜	* : ♥ ー ゛
3	J ミ	I │ フ	H 時 マ	G ┤ ソ	F ┼ セ	E ┌ ク	D ├ ス	C └ ッ ツ
4	R ┤ ケ	Q カ	P π ホ	O ヘ	N ャ ヤ	M 分 ユ	L 中 メ	K ム
5	Z タ	Y 年 ハ	X × チ	W キ	V ┴ テ	U ヒ	T ┐ コ	S 秒 シ
6	F8 F3	F7 F2	F6 F1	かな	CAPS LOCK	GRAPH	CTRL	SHIFT
7	RETURN	SELECT	BS	STOP	TAB	ESC	F10 F5	F9 F4
8	→	↓	↑	←	DEL	INS	CLS HOME	SPACE

[TEN KEY]

行	7	6	5	4	3	2	1	0
9	4	3	2	1	0	option	option	option
10	.	,	−	9	8	7	6	5

図5.16 キーマトリクス

リスト5.4　キースキャンルーチンの使用法

```
;==========================================================
;
;  List 5.4   scan key-matrix and display it
;
;==========================================================
;
CHPUT   EQU     00A2H
BREAKX  EQU     00B7H
POSIT   EQU     00C6H
SNSMAT  EQU     0141H

        ORG     0B000H

;----- program start -----          キーマトリクスを読んでそのビットパターンを表示する

SCAN:   LD      C,0                 ;C := line of key matrix

SC1:    LD      A,C
        CALL    SNSMAT              ;Read key matrix

        LD      B,8
        LD      HL,BUF              ;HL := buffer address
SC2:    LD      D,'.'
        RLA                         ;Check bit
        JR      C,SC3
        LD      D,'#'
SC3:    LD      (HL),D              ;store '.' or '#' to buffer
        INC     HL
        DJNZ    SC2

        LD      H,05H               ;x := 5
        LD      L,C                 ;y := C+1
        INC     L
        CALL    POSIT               ;set cursor position

        LD      B,8                 ;put out bit patterns to CRT
        LD      HL,BUF
SC4:    LD      A,(HL)
        CALL    CHPUT
        INC     HL
        DJNZ    SC4

        CALL    BREAKX              ;check Ctrl-STOP
        RET     C

        INC     C                   ;line No. increment
        LD      A,C
        CP      09
        JR      NZ,SC1
        JR      START

;----- work area -----

BUF:    DS      8

        END
```

3.3 文字の入力

MSXは，タイマ割り込みにより1/60秒ごとにキーマトリクスをスキャンしており，キーが押されていれば，その文字コードを**図5.17**のようなリング状のキーボードバッファに格納します．なお，MSXのキー入力は，一般にこのキーボードバッファを読むことによって行っています．

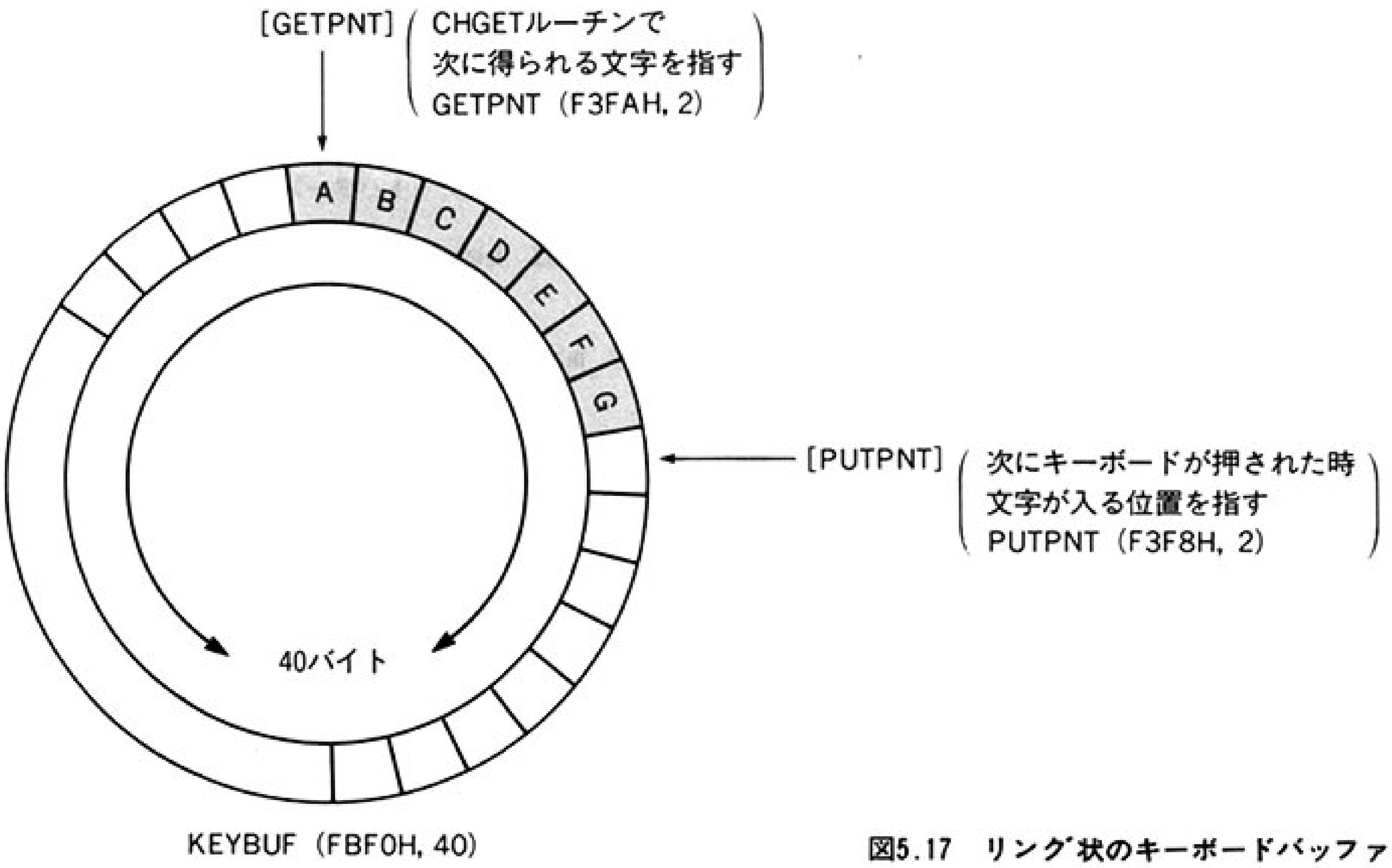


図5.17　リング状のキーボードバッファ

このキーボードバッファを使用したキー入力およびそれに関連する機能を持ったBIOSを以下に示します．タイマ割り込みを禁止すると，当然これらは使用できなくなります．

● CHSNS (009CH/MAIN)……………キーボードバッファの残りのチェック

入力：――

出力：バッファが空なら，Zフラグ＝ON

機能：キーボードバッファに文字が残っているかどうかを調べる．キーボードバッファが空であればZフラグを立てる．

● CHGET (009FH/MAIN)……………キーボードバッファから1文字入力

入力：――

出力：A←文字コード

機能：キーボードバッファから1文字読み出してAレジスタに格納する．バッファに文字が入っていない場合には，カーソルを表示してキー入力があるまで待つ．キー入力待ちの間，

CAP ロック，カナロック，ローマ字カナ変換ロックも有効である．関係のあるワークエリアは以下に示すとおりである．このうち，SCNCNT と REPCNT は CHGET ルーチン実行後に初期化されてしまうので，オートリピートの時間間隔を変えるためには CHGET コールのたびにこのワークを設定する必要がある．

ワークエリア

CLIKSW	(F3DBH ,1)	キークリック音（0＝OFF, 0 以外＝ON）
SCNCNT	(F3F6H ,1)	キースキャンの時間間隔（通常は 1）
REPCNT	(F3F7H ,1)	オートリピート開始までの時間（通常は50）
MODE	(FAFCH ,1)	ローマ字カナ変換のモード（図 5.18 参照）
CSTYLE	(FCAAH ,1)	カーソルの形（0＝■　0 以外＝▬ ）
CAPST	(FCABH ,1)	CAPS ロック（0＝OFF, 0 以外＝ON）
KANAST	(FCACH ,1)	カナロック（0＝OFF, 0 以外＝ON）

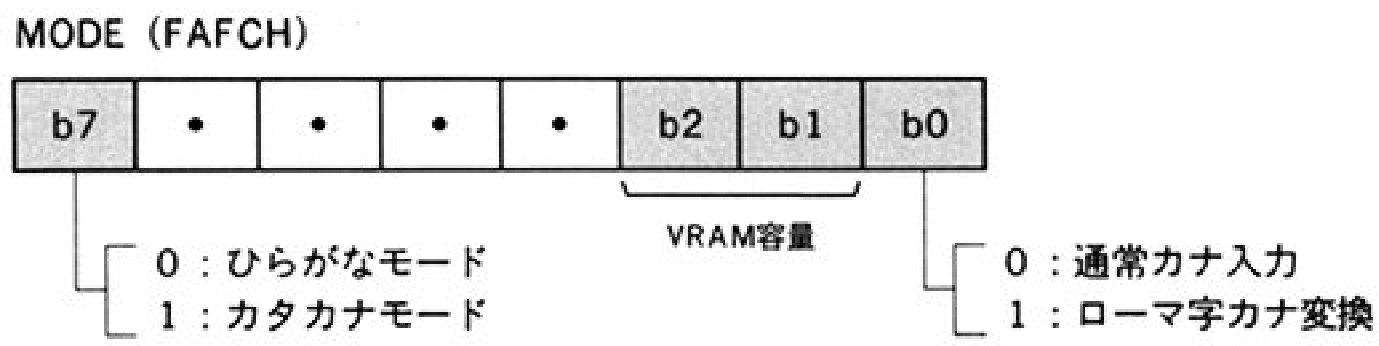


図5.18　ローマ字カナ変換モードの設定

● KILBUF (0156H／MAIN) ……………キーボードバッファを空にする

入力：――

出力：――

機能：キーボードバッファを空にする．

リスト5.5　1 文字入力ルーチンの使用法

```
;=========================================================
;
;   List 5.5     get key code
;
;                this routine dosen't wait key hit
;
;=========================================================
;
CHSNS    EQU      009CH           ;check keyboard buffer
CHGET    EQU      009FH           ;get a character from buffer
CHPUT    EQU      00A2H           ;put a character to screen
BREAKX   EQU      00B7H           ;check Ctrl-STOP
KILBUF   EQU      0156H           ;clear keyboard buffer
```

```
REPCNT  EQU     0F3F7H          ;time interval until key-repeat
KEYBUF  EQU     0FBF0H          ;keyboard buffer address

        ORG     0B000H

;----- program start -----      CHGETを使ったリアルタイムキー入力

KEY:    CALL    CHSNS           ;check keyboard buffer
        JR      C,KEY1

        LD      A,1
        LD      (PEPCNT),1      ;not to wait until repeat
        CALL    CHGET           ;get a character (if exists)
        JR      KEY2

KEY1:   LD      A,'-'           ;A := '-'

KEY2:   CALL    CHPUT           ;put the character
        CALL    KILBUF          ;clear keyboard buffer
        CALL    BREAKX          ;check Ctrl-STOP
        JR      NC,KEY

        END
```

● CNVCHR（00ABH／MAIN）…………グラフィック文字の処理

入力：A←文字コード

出力：A←グラフィック文字は変換される（通常文字ならば変換されない）

CY フラグ＝OFF　（入力はグラフィックヘッダバイト01Hだった）

CY フラグ＝ON，Z フラグ＝ON　（入力はグラフィック文字なので変換された）

CY フラグ＝ON，Z フラグ＝OFF（入力は通常の文字なので変換されなかった）

機能：CHGET の後にこの CNVCHR を実行すると，グラフィック文字は図5.19のような１バイトコードに変換し，グラフィック文字以外は無変換でそのまま返す．グラフィック文字はグラフィックヘッダバイト（01H）を伴う２バイトの変則的なコードで表されるため文字処理の時にめんどうな手続きが必要だが，このルーチンによって多少は手間が省ける．

● PINLIN（00AEH／MAIN）……………１行入力

入力：――

出力：HL ← F55DH

[F55E] ←入力した文字列（行末は00Hで示される）

CY フラグ← STOP で終了した場合＝ON，RETURN で終了した場合＝OFF

変換前	後	変換前	後
		0150H (π) →	50H
0141H (月) →	41H	0151H (┴) →	51H
0142H (火) →	42H	0152H (┬) →	52H
0143H (水) →	43H	0153H (┤) →	53H
0144H (木) →	44H	0154H (├) →	54H
0145H (金) →	45H	0155H (┼) →	55H
0146H (土) →	46H	0156H (│) →	56H
0147H (日) →	47H	0157H (─) →	57H
0148H (年) →	48H	0158H (┌) →	58H
0149H (円) →	49H	0159H (┐) →	59H
014AH (時) →	4AH	015AH (└) →	5AH
014BH (分) →	4BH	015BH (┘) →	5BH
014CH (秒) →	4CH	015CH (□) →	5CH
014DH (百) →	4DH	015DH (大) →	5DH
014EH (千) →	4EH	015EH (中) →	5EH
014FH (万) →	4FH	015FH (小) →	5FH

図5.19　グラフィック文字のコード変換表

機能：入力した文字列をラインバッファ BUF (F55DH) に格納する．文字列の入力時にはスクリーンエディットのすべての機能が有効である．RETURN キーまたは STOP キーを押すと入力動作を終了する．なお，ワークエリアは以下に示すとおりである．

ワークエリア

BUF	(F55DH, 258)	文字列が格納されるラインバッファ
LINTTB	(FBB2H, 24)	物理的1行が，上の行の継続であれば00H

● INLINE (00B1H／MAIN) ……………1行入力（プロンプト使用可）

入力：——

出力：PINLIN と同じ

機能：PINLIN ルーチンと同様に，入力した文字列をラインバッファ BUF (F55DH) に格納する．ただし，こちらはルーチン実行開始時のカーソル位置より前の部分は入力されない．両者の違いを**リスト5.6**に示す．

リスト5.6　INLINとPINLINの違い

```
;==========================================================
;
;  List 5.6   INLIN and PINLIN
;
;==========================================================
;
CHPUT   EQU     00A2H
INLIN   EQU     00B1H
PINLIN  EQU     00AEH
KILBUF  EQU     0156H

BUF     EQU     0F55EH

        ORG     0B000H

;----- program start -----

        LD      HL,PRMPT1
        CALL    PUTMSG          ;put prompt message
        CALL    INLIN           ;use INLIN routine
        LD      HL,BUF
        CALL    PUTMSG

        LD      HL,PRMPT2
        CALL    PUTMSG          ;put prompt message
        CALL    PINLIN          ;use PINLIN routine
        LD      HL,BUF
        CALL    PUTMSG

        RET

;----- put a string -----

PUTMSG: LD      A,(HL)
        CP      '$'
        RET     Z
        CALL    CHPUT
        INC     HL
        JR      PUTMSG

;----- string data -----

PRMPT1: DB      0DH,0AH,'INLIN:$'
PRMPT2: DB      0DH,0AH,'PINLIN:$'

        END
```

3.4　ファンクションキー

MSXには10個のファンクションキーが存在し，ユーザーが自由に定義して使うことができます．ファンクションキーの定義を行うために，ワークエリアが各キーについて16バイトずつ割り当てられています．そのアドレスを以下に示します．

FNKSTR (F87FH, 16) ……　F・1 キーの定義用領域
+10H (F88FH, 16) ……　F・2 キーの定義用領域
+20H (F89FH, 16) ……　F・3 キーの定義用領域
+30H (F8AFH, 16) ……　F・4 キーの定義用領域
+40H (F8BFH, 16) ……　F・5 キーの定義用領域
+50H (F8CFH, 16) ……　F・6 キーの定義用領域
+60H (F8DFH, 16) ……　F・7 キーの定義用領域
+70H (F8EFH, 16) ……　F・8 キーの定義用領域
+80H (F8FFH, 16) ……　F・9 キーの定義用領域
+90H (F90FH, 16) ……　F・10 キーの定義用領域

1つのファンクションキーを押すと，それぞれの領域に定義された文字列が［KEYBUF］に格納されます．文字列の最後は00Hで示され，最大15文字まで定義可能です（16文字を超えた分は，複数のファンクションキー定義領域にまたがって定義される）．ファンクションキーを初期設定の状態に戻すには，次のBIOSルーチンを利用するとよいでしょう．

● **INIFNK (003EH／MAIN)** ……ファンクションキーの初期化

入力：——

出力：——

機能：ファンクションキーの定義をBASICスタート時の設定に戻す．

3.5　割り込み中のSTOPキー

3.3で説明した1文字入力ルーチンCHGETは，どのキーが押されているかの判断をタイマ割り込みの処理ルーチン内で行っています．したがって，たとえばカセットデータの入出力時などタイマ割り込みが禁止されている状態では，どんなキーが押されていても読み取ることができません．しかし，次に述べるBIOSルーチンを使用すると，割り込みが禁止されている場合でも，CTRLキー＋STOPキーが押されていることを判定できます．

● BREAKX (00B7H／MAIN) ……………CTRL+STOPの判定

入力：——

出力：CTRL+STOPが押されていれば，CYフラグ=ON

機能：キースキャンを行い，CTRLキーとSTOPキーが同時に押されているか否かを判定する．もし両キーが押されていれば，CYフラグを“1”にセットして戻る．CTRL+STOPが押されていなければ，CYフラグを“0”にリセットして戻る．このルーチンは，割り込み禁止中でも使用可能．

4 章　プリンタ・インターフェイス

本章では，MSX のプリンタ・インターフェイスをマシン語からアクセスする方法について説明します．ここで述べる情報は，特にビットイメージ印字によってグラフィックスを表示するような目的でプリンタを使用する場合に必要となってきます．

4.1　プリンタ・インターフェイスの概要

BIOS と BASIC により，プリンタのサポートをしているので説明します．MSX は規格により 8 ビットパラレルの出力ポートで，BUSY と STROBE 信号によるハンドシェーク方式でプリンタを動かします．コネクタなども規格に定められています(アンフェノール14ピン，本体側メス)．信号線表を図 5.20 に示します．

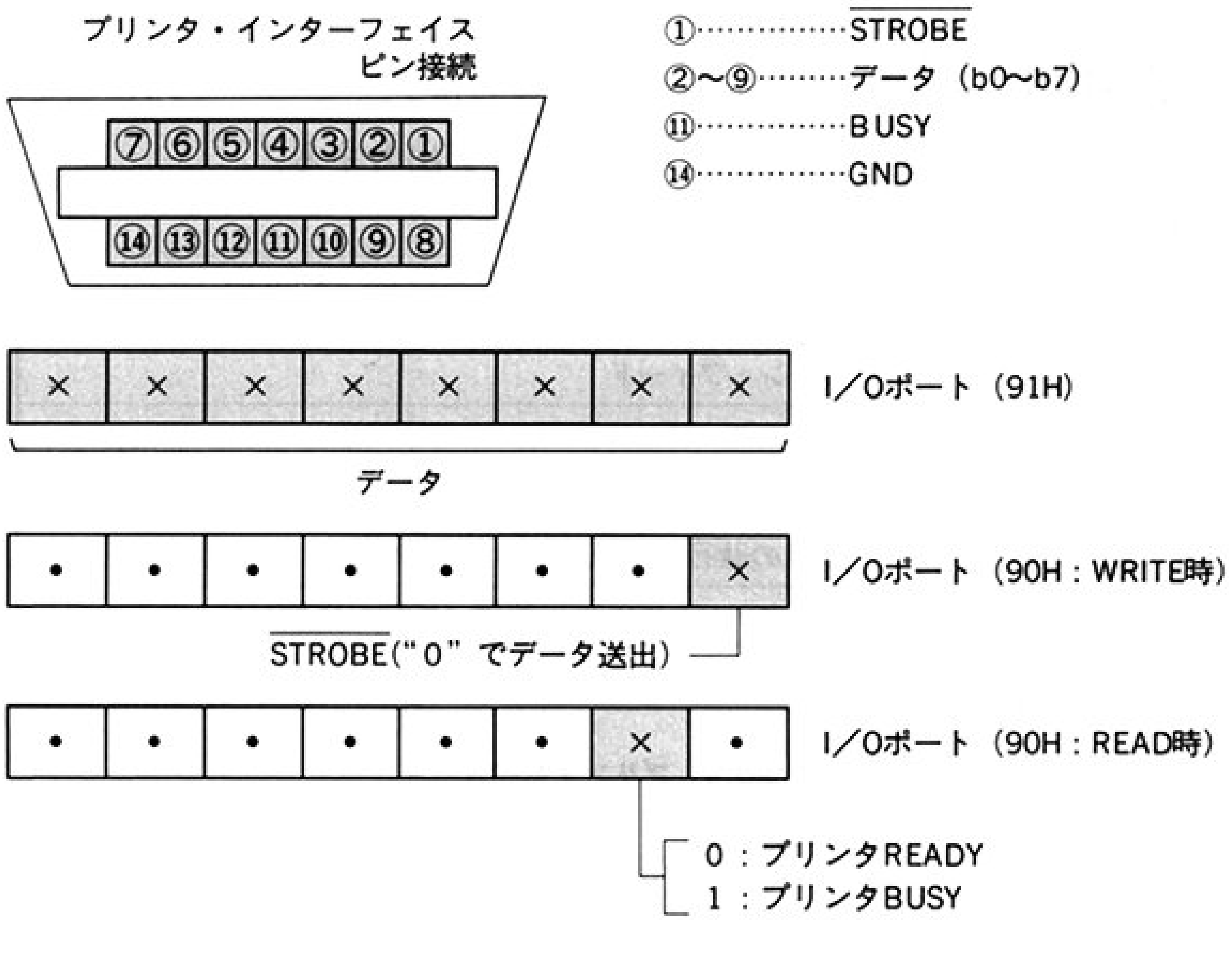


図5.20　プリンタ・インターフェイス

4.2 MSX仕様のプリンタへの出力

MSXからプリンタへデータを送る場合，送られる側のプリンタがMSX仕様であるか否かによって，その動作は異なります．そこで，まずMSX仕様のプリンタの使い方について説明しましょう．MSX仕様以外のプリンタについては次の節で述べることにします．

MSX仕様のプリンタは，MSXが画面に表示できるすべてのキャラクタが印字でき，キャラクタコードn=01H～1FHに相当するグラフィックキャラクタ（月，火，水……）も，グラフィック・キャラクタ・ヘッダ（01H）の後に40H+nというコードを出力することによって印字可能です．さらに，MSX仕様のプリンタは，少なくとも，**表5.4**に示す制御コードが使用できます（漢字印字など，これ以外の機能を備えたプリンタの制御に関しては，それぞれのプリンタの説明書を参照）．

MSX仕様のプリンタに改行させるためには，0DHと0AHを続けて出力してください（MSX仕様でないプリンタには，0DHのみで改行する機種もある）．ビット・イメージ印字を行うには，ESC+"Snnnn"（nnnnは10進4桁の数字）というエスケープ・シーケンスの後に，nnnnバイトのデータを出力します．ただし，MSXはタブ機能を持たないプリンタのために，タブコード（09H）を適当な数のスペースコード（20H）に変換してプリンタに送る機能があり，通常は常にこの変換を行ってしまいます．09Hという値を含むビット・イメージを正しく表示するためには，次のワークエリアを書き換えてください．

● RAWPRT（F418H, 1）　………内容が00Hならばタブをスペースに置き換え，00H以外ならばそのまま出力する．

コード	機　　能
0AH	ライン・フィード
0CH	フォーム・フィード
0DH	キャリッジ・リターン
ESC+"A"	通常の改行間隔（文字が読みやすいように行間を空ける）
ESC+"B"	グラフィック用の改行間隔（行間の隙間がなくなる）
ESC+"Snnnn"	ビット・イメージ印字

表5.4　プリンタ制御コード

4.3　MSX仕様以外のプリンタへの出力

MSX仕様ではないプリンタを使用する場合に問題となるのは，“ひらがな”をどう扱うかという点でしょう．普通は，カタカナは印字できても，ひらがなは印字できないというプリンタが一般的です．MSXには，そのようなプリンタのために，ひらがなをカタカナに変換して出力する機能があります．BASICからは，SCREEN命令の第5パラメータで指定しますが，これは次のワークエリアを書き換えることによっても可能です．

● NTMSXP(F417H,1) ………………… 内容が00Hならば，ひらがなをカタカナに変換，00H以外ならば，ひらがなをそのまま出力．

4.4　プリンタのアクセス

プリンタへの出力を行うために，以下に示すBIOSルーチンが用意されています．

● LPOUT (00A5H／MAIN)

入力：Aレジスタ←文字コード

出力：異常終了時CYフラグ＝ON

機能：Aレジスタで指定した文字をプリンタに出力する．

● LPTSTT (00A8H／MAIN)

入力：———

出力：Aレジスタ←プリンタの状態

機能：現在のプリンタの状態をチェックする．このルーチンを呼び出して，Aレジスタが255でかつZフラグが0ならばプリンタは使用可能であり，Aレジスタが0でかつZフラグが1ならば使用できない．

● OUTDLP（014DH／MAIN）

入力：Aレジスタ←文字コード

出力：異常終了時CYフラグ＝ON

機能：Aレジスタで指定した文字をプリンタに出力する．LPTOUTルーチンとの相違点は次のとおりである．

・TABコードは相当する個数のスペースをプリントする．

・MSX仕様でないプリンタでひらがなを出力する場合，カタカナに変換する．

・異常終了した場合，Device I／O errorとなって返ってくる．

5章 汎用入出力インターフェイス

1章でも説明したように，MSXの使用しているPSGは，音声出力の機能とは別にポートAとポートBという2つの8ビット入出力ポートを持っています．MSXはこの2つのポートを汎用入出力インターフェイス（いわゆるジョイスティックポート）に接続して，ジョイスティックやパドルなどの装置とのデータ入出力に利用しています（**図5.21**）．この汎用入出力インターフェイスに接続される各種の装置は，それぞれ専用のBIOSルーチンがROM内に用意されており，手軽にアクセスすることが可能です．

本章では，各入出力装置の機能とBIOSルーチンによるアクセス法について説明します．

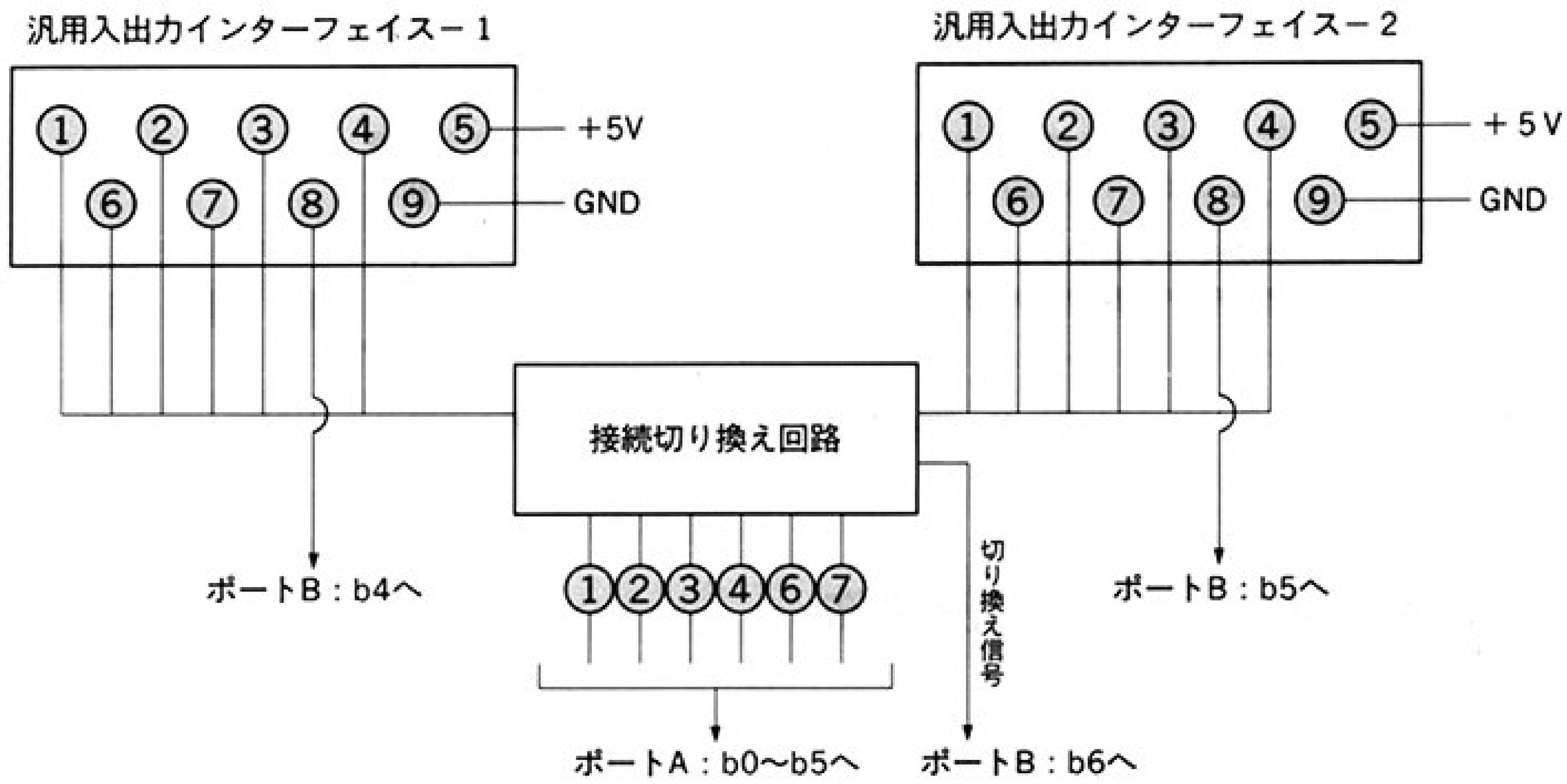


図5.21　汎用入出力インターフェイス

5.1　ポートの機能

PSG の２つの入出力ポートは，図 5.22 のように利用されています．

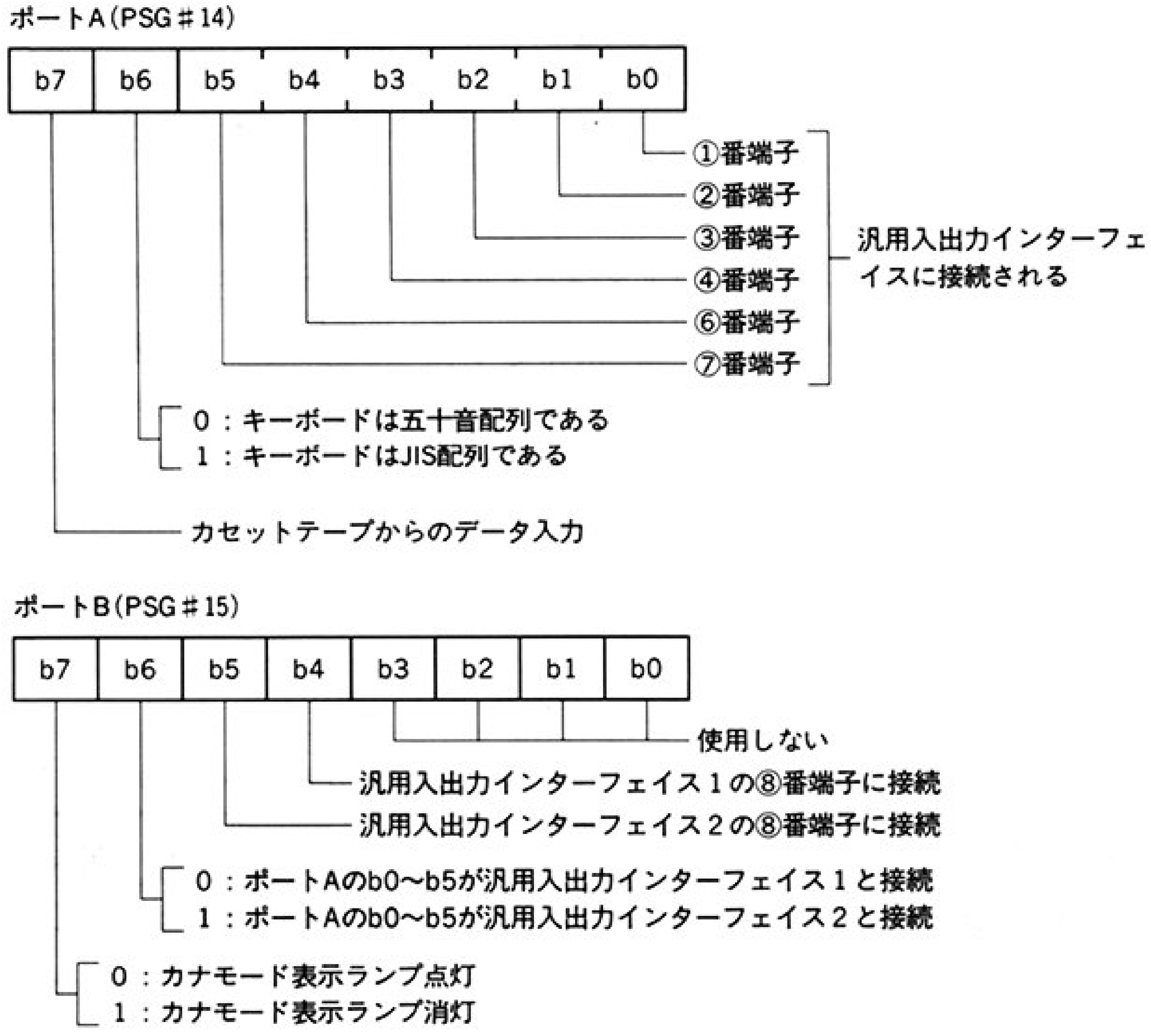


図5.22　PSGのポートAとポートBの機能

5.2　ジョイスティックの使用法

ジョイスティックの回路を図 5.23 に示します．この回路からわかるように，⑧番端子に“0”を出力し，①～④，⑥～⑦の端子を読み出せば，スティックとトリガボタンの情報は得られますが，プログラムの移植性などを考慮すると，ジョイスティックのアクセスはやはり BIOS を利用して行う方が無難でしょう．

ジョイスティックをアクセスするためには，以下に示す BIOS ルーチンが用意されています．なお，これらのルーチンは BASIC の STICK 関数や STRIG 関数とほぼ等しい機能を持っているもので，ジョイスティック以外にカーソルキーやスペースバーの状態をリアルタイムで読み取ることも可能です．

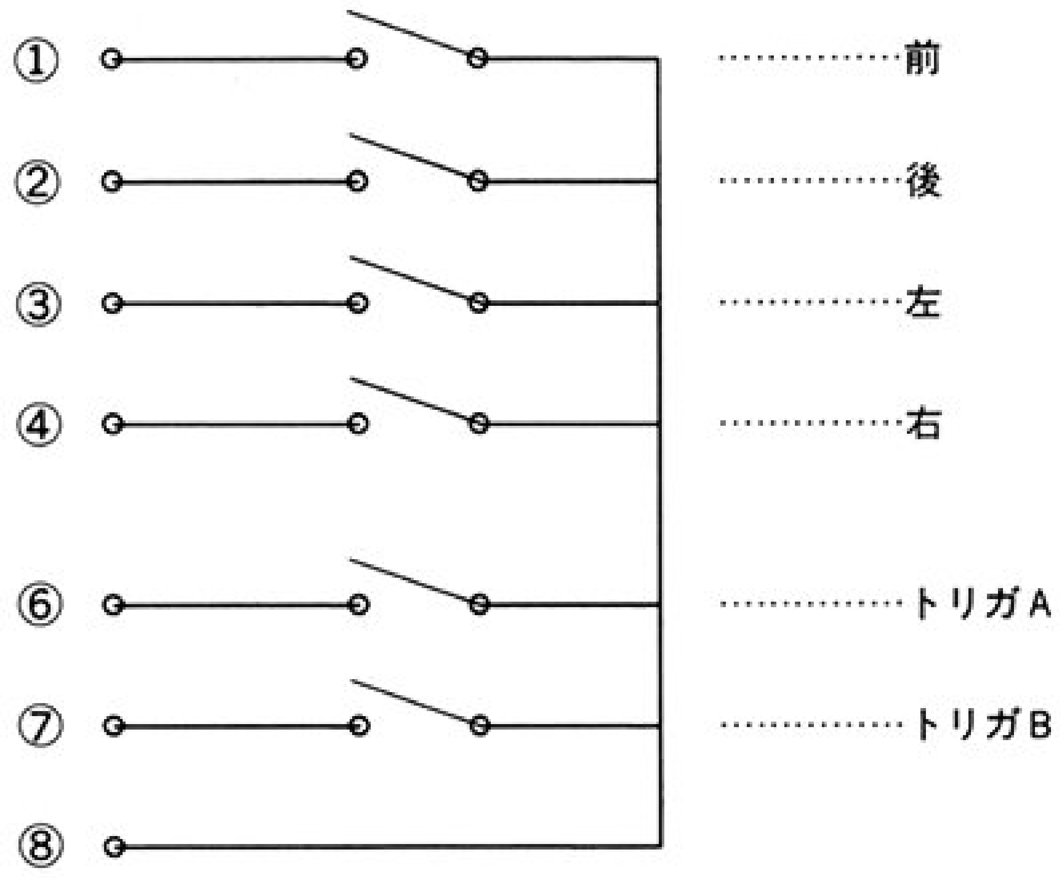


図5.23　ジョイスティック回路図

● GTSTCK（00D5H／MAIN）……………ジョイスティックの読み出し

入力：A←ジョイスティック番号（0＝カーソルキー，1～2＝ジョイスティック）

出力：A←ジョイスティックまたはカーソルキーの押された方向

機能：現在のジョイスティックまたはカーソルキーの状態をAレジスタに返す．値はBASICのSTICK関数と同じである．

● GTTRIG（00D8H／MAIN）……………トリガボタンの読み出し

入力：A←トリガボタン番号（0＝スペースバー，1～2＝トリガボタン）

出力：A←トリガボタンまたはスペースバーの状態（押す＝0FFH，離す＝00H）

機能：現在のトリガボタン，またはスペースバーの状態をAレジスタに返す．この値はトリガが押されていれば0FFH，押されていなければ0となる．

リスト5.7　ジョイスティックの使用法

```
;=============================================================
;
; List 5.7  Joystick and trigger accsess
;
;=============================================================
;
CHPUT   EQU     00A2H
BREAKX  EQU     00B7H
GTSTCK  EQU     00D5H
GTTRIG  EQU     00D8H

        ORG     0D000H
```

```
;-------- program start --------ジョイスティックの状態を表示する

STICK:  LD      A,1                 ;choose joystick 1
        CALL    GTSTCK              ;read joystick status.
        LD      (WK1),A
        LD      A,1                 ;choose joystick 1
        CALL    GTTRIG              ;read trigger status.

        OR      A
        JR      Z,STCK1
        LD      HL,WDON             ;trigger ON
        JR      STCK2
STCK1:  LD      HL,WDOFF            ;trigger OFF
STCK2:  CALL    PUTSTR
        LD      A,(WK1)
        OR      A
        JR      Z,BRKCH0            ;no use joystick.
        LD      C,0
STCK3:  DEC     A
        JR      Z,STCK4
        INC     C
        JR      STCK3

STCK4:  SLA     C                   ;C := C*16
        SLA     C
        SLA     C
        SLA     C
        LD      B,0                 ;Accounting Strings data address.
        LD      HL,WDSTK
        ADD     HL,BC
        CALL    PUTSTR

BRKCH0: LD      A,0DH               ;put carriage return
        CALL    CHPUT               ;code := 0DH

BRKCHK: CALL    BREAKX              ;break check
        RET     C
        JR      STICK

;----- put strings to screen -----

PUTSTR: LD      A,(HL)
        CP      '$'
        RET     Z
        INC     HL
        CALL    CHPUT
        JR      PUTSTR

;----- string data -----

WDON:   DB      'Trigger ON:  $'
WDOFF:  DB      'Trigger OFF: $'
WDSTK:  DB      'UP only       ',0DH,0AH,'$'
        DB      'Up and Right  ',0DH,0AH,'$'
        DB      'Right only    ',0DH,0AH,'$'
        DB      'Right & Down  ',0DH,0AH,'$'
        DB      'Down olny     ',0DH,0AH,'$'
        DB      'Down and Left',0DH,0AH,'$'
        DB      'Left only     ',0DH,0AH,'$'
        DB      'Left and Up   ',0DH,0AH,'$'

WK1:    DW      0

        END
```

5.3 パドルの使用法

パドルの回路は図5.24のとおりです．⑧端子にパルスを送ると単安定マルチバイブレータは，ある時間幅のパルスを発生します．この幅は可変抵抗の値によって10μs～3 msの範囲で変化しますので，パルス長を測定すれば可変抵抗の値，ひいてはその回転角を知ることができます．

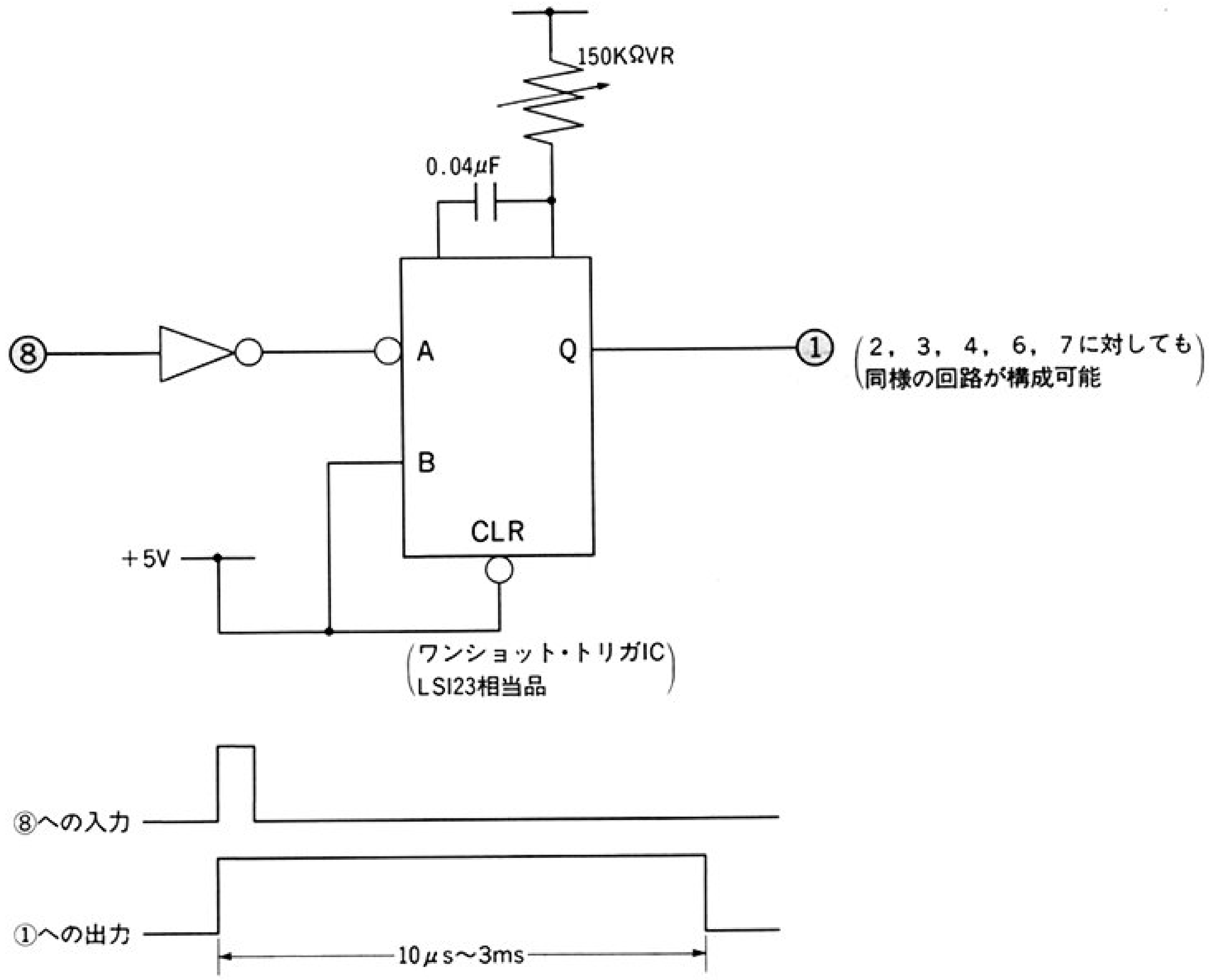


図5.24　パドル回路図

パドルをアクセスするためのBIOSルーチンを次に示します．

● GTPDL（00DEH／MAIN）……………パドル情報の読み出し

入力：A←パドル番号（1～12）

出力：A←パドルの回転角（0～255）

機能：Aレジスタで指定したパドルの状態を調べ，結果をAレジスタに返す．

5.4　タッチパネル，ライトペン，マウス，トラックボールの使用法

タッチパネル，ライトペン，マウス，トラックボール（キャット）はすべて同一のBIOSを用いてアクセスすることが可能です．その使い方を以下に説明します．

● GTPAD（00DBH／MAIN）……………各種入出力装置のアクセス

入力：A←装置ID（0～19）

出力：A←目的の情報

機能：Aレジスタで指定する値によって，**表5.5**のような各種の情報を得ることができる．これはBASICのPAD関数とまったく同じである．表中“XXX1”は，汎用入出力インターフェイス#1に接続した“XXX”の装置であることを意味する．同様に“XXX2”は汎用入出力インターフェイス#2に接続したものである．

装置ID	指定される装置	返ってくる情報
0	タッチパネル1	パネル面に触っていれば0FFH，いなければ00H
1		X座標(0～255)
2		Y座標(0～255)
3		ボタンが押されていれば0FFH，いなければ00H
4 5 6 7	タッチパネル2	同　上
8	ライトペン	0FFHであればデータは有効，00Hであれば無効
9		X座標(0～255)
10		Y座標(0～255)
11		スイッチが押されていれば0FFH，いなければ00H
12	マウス1，または トラックボール1	つねに0FFHを返す(入力の要求に使用)
13		X座標(0～255)
14		Y座標(0～255)
15		つねに00Hを返す(無意味)
16 17 18 19	マウス2，または トラックボール2	同　上

注1：ライトペンの座標(A=9，10)とスイッチ(A=11)の情報は，A=8としてBIOSをコールしたとき同時に読み込まれますが，その結果が0FFHの時にのみ，他の値が有効となります．A=8としてBIOSをコールした結果が00Hの場合は，その後に得られる座標値やスイッチの状態は意味を持ちません．

注2：マウスとトラックボールは自動的に判別します．

注3：マウスやトラックボールの座標値を求める場合，まず入力要求のコール(A=12またはA=16)を行い，その後に実際に座標値を求めるコールを実行する，という手順をふみます．このとき2つのコールの時間間隔はできる限り小さくとらなければなりません．入力要求から座標値の入力までに必要以上に時間をかけると，得られたデータは保証されません．

注4：マウスまたはトラックボールについているトリガボタンの状態を得るためには，このGTPADルーチンではなく，GTTRIG(00D8H／MAIN)を使ってください．

表5.5　GTPADのBIOSで得られる情報

リスト5.8　タッチパネルの使用法

```
;================================================================
;
;  List 5.8    touch pad access
;
;================================================================
;
BREAKX  EQU     00B7H
GTPAD   EQU     00DBH
WRTVRM  EQU     004DH

        ORG     0B000H

;----- program start -----           タッチパッドで指定した位置に“*”を表示する

PAD:    XOR     A                    ;check sense
        CALL    GTPAD
        OR      A
        JR      NZ,PAD1
        LD      A,3
        CALL    GTPAD                ;break check
        OR      A
        RET     NZ
        JR      PAD

PAD1:   LD      A,1                  ;get X axis
        CALL    GTPAD
        SRL     A                    ;A := A/8
        SRL     A
        SRL     A
        LD      (WORK),A             ;reserve X axis
        LD      A,2                  ;get Y axis
        CALL    GTPAD
        LD      L,A                  ;HL := Y data(0-255)
        LD      H,0
        LD      C,A
        LD      B,0
        ADD     HL,BC                ;HL := HL*3  (HL:= 0-767)
        ADD     HL,BC
        LD      A,L
        AND     11100000B
        LD      L,A
        LD      A,(WORK)
        ADD     A,L
        LD      L,A
        LD      BC,1800H             ;VRAM start address.
        ADD     HL,BC
        LD      A,2AH
        CALL    WRTVRM               ;write VRAM.
        LD      A,3
        CALL    GTPAD                ;break check.
        OR      A
        RET     NZ
        JR      PAD

;----- work area -----

WORK:   DW      0                    ;work

        END
```

リスト5.9 マウスとトラックボールの使用法

```
;============================================================
;
;   List 5.9   mouse and track ball accsess
;
;============================================================
;
GTPAD   EQU     00DBH
WRTVRM  EQU     004DH
RDVRM   EQU     004AH
BREAKX  EQU     00B7H

        ORG     0D000H

;----- program start -----          マウスまたはトラックボールで指定した位置に"*"を表示する

TEST:   CALL    VADR                ;Put old data.
        LD      A,(WKOLD)
        CALL    WRTVRM
        LD      A,12
        CALL    GTPAD               ;Request mouse/track ball data.
        LD      A,13
        CALL    GTPAD               ;Read X val.
        LD      (WKXVAL),A
        LD      A,14
        CALL    GTPAD               ;Read Y val.
        LD      (WKYVAL),A

        LD      A,(WKX)
        LD      B,A
        LD      A,(WKXVAL)
        ADD     A,B
        CP      245                 ;X<0?
        JR      C,TEST01
        XOR     A                   ;X=0
        JR      TEST02

TEST01: CP      32                  ;X>31?
        JR      C,TEST02
        LD      A,31

TEST02: LD      (WKX),A

        LD      A,(WKY)
        LD      B,A
        LD      A,(WKYVAL)
        ADD     A,B
        CP      245                 ;Y<0?
        JR      C,TEST03
        XOR     A                   ;Y=0
        JR      TEST04

TEST03: CP      24                  ;Y>23
        JR      C,TEST04
        LD      A,23

TEST04: LD      (WKY),A

        CALL    VADR
        CALL    RDVRM               ;Read old data.
        LD      (WKOLD),A

        CALL    VADR
        LD      A,2AH
        CALL    WRTVRM              ;Put cursor("*").
```

```
        CALL    BREAKX          ;Break check.
        RET     C

        CALL    WAIT

        JR      TEST

VADR:   LD      A,(WKY)         ;Make SCREEN Address.
        LD      H,A             ; From X,Y axis on WORK ARIA
        LD      L,0             ; To HL reg.
        SRL     H
        RR      L
        SRL     H
        RR      L
        SRL     H
        RR      L
        LD      A,(WKX)
        ADD     A,L             ; Y*32+X
        LD      L,A
        LD      BC,1800H        ; VRAM start address.
        ADD     HL,BC
        RET

WAIT:   LD      A,0             ;WAIT routine
WLP1:   INC     A
        LD      B,(IX+0)
        LD      B,(IX+0)
        LD      B,(IX+0)
        JR      NZ,WLP1
        RET

;----- data -----

WKX:    DB      10              ;X axis
WKY:    DB      10              ;Y axis
WKOLD:  DB      0               ;Character code on (X,Y)
WKXVAL: DB      0               ;X variable.
WKYVAL: DB      0               ;Y variable.

        END
```

6章 CLOCKとバッテリバックアップ・メモリ

MSX 2はCLOCK-ICを用いて時計機能を実現しています．このICはバッテリバックアップされており，MSX 2本体の電源を切っても動作を続けるようになっています．またそのために内蔵されている少量のRAMを，MSX 2ではCLOCK機能のほかにPASSWORDの設定やスタート時のスクリーンモードの自動設定などに利用しています．

6.1 CLOCK-ICの機能

このICの機能は以下の3つに分けられます．

● CLOCK機能

・「年，月，日，曜日，時，分，秒」の設定／読み出しができる．

・時刻の表現は，24時間計／12時間計の切り換えができる．

・月の更新は，大の月と小の月を考慮する（4年に1度の閏年も判別する）．

● アラーム機能

・アラーム時刻を設定しておくと，CLOCKがその値に一致した時点で信号を発生する．

・アラーム時刻は「XX日　XX時　XX分」の単位で設定できる．

● バッテリバックアップ・メモリ機能

・26個の4ビットメモリを持ち，各種の情報をバッテリバックアップすることができる．

・MSX 2では，このメモリに以下のようなデータを記憶させている．

1. CRT表示の上下左右の調整値
2. SCREEN，WIDTH，COLORの初期設定値
3. BEEPの音色と音量
4. タイトル画面の色
5. 国別コード
6. パスワード ┐
7. BASICのプロンプト ├ どれかひとつが有効
8. タイトル文字 ┘

6.2　CLOCK-ICの構造

CLOCK-ICの内部は，図5.25に示すように4つのブロックに分かれています．各ブロックは13個の4ビットレジスタで構成され，それぞれのブロック内のレジスタは0〜12のアドレスで指定します．またブロックの選択や機能コントロール用に3個の4ビットレジスタを持ち，これを13〜15のアドレスで指定します．

各ブロックに含まれるレジスタ（#0〜#12）と，MODEレジスタ（#13）は書き込みも読み出しも可能です．TESTレジスタ（#14）とRESETレジスタ（#15）は書き込み専用で読み出しはできません．

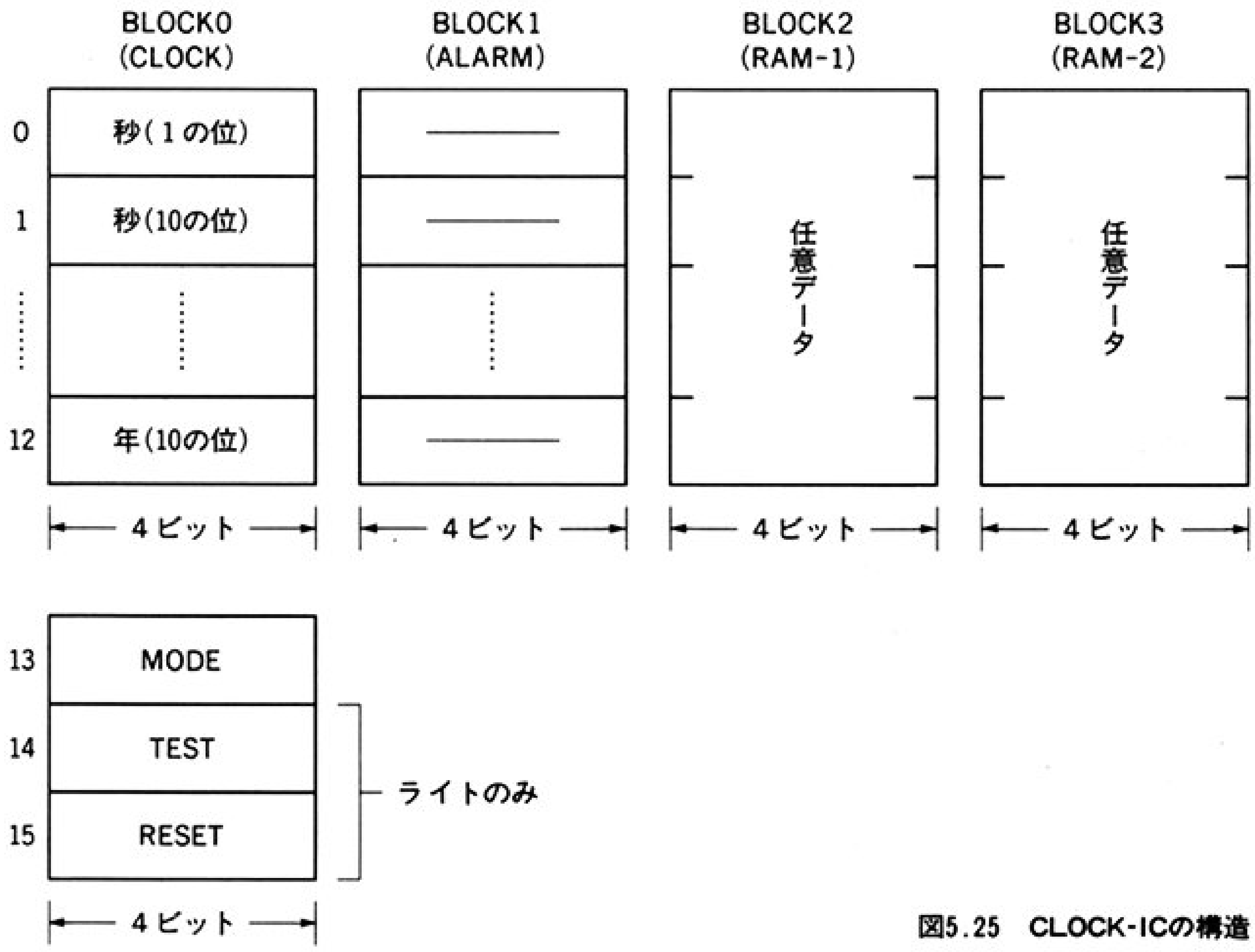


図5.25　CLOCK-ICの構造

6.3　MODEレジスタの機能

MODEレジスタ（#13）は以下の3つの機能を持ちます．

●ブロックの選択

#0〜#12のレジスタの読み書きを行う場合は，まず使用するブロックを選択してから目的のアドレスをアクセスしなければなりません．ブロックの選択にはMODEレジスタの下位2ビット

を使用します．

なお，#13～#15のレジスタは，どのブロックが選択されている時でもアクセス可能です．

● アラーム出力のON／OFF

MODEレジスタのビット2でアラーム出力のON／OFFを行います．ただしMSX 2の標準仕様ではアラームに関するサポートはしていませんから，このビットを書き換えても一般には何も起こりません．

● CLOCKカウントの停止

MODEレジスタのビット3を“0”にすると，秒以降のカウントを停止し（秒より前の分周段は止まらない），時計機能をストップさせます．ビット3を“1”にするとカウントを再開します．

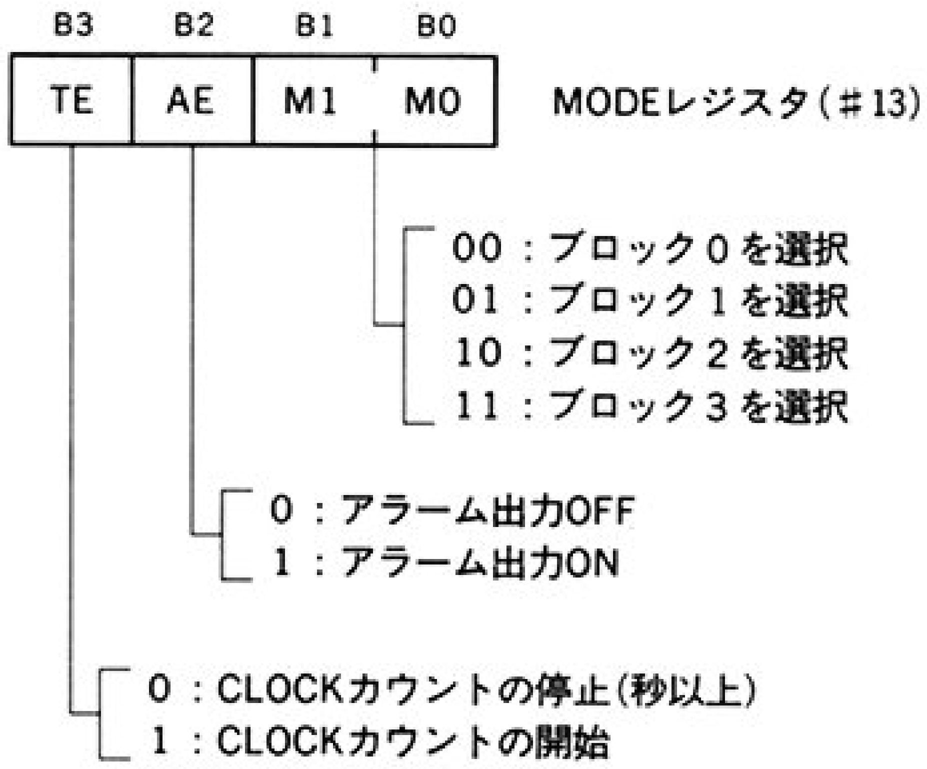


図5.26　MODEレジスタの機能

6.4　TESTレジスタの機能

TESTレジスタ（#14）は上位のカウンタを素早くカウントアップさせ，時刻や日付の繰り上がり動作を確認するために用います．このレジスタの各ビットに“1”を立てると，それぞれ日，時，分，秒のカウンタに直接2^{14}（=16384）［Hz］のパルスが入力されます．

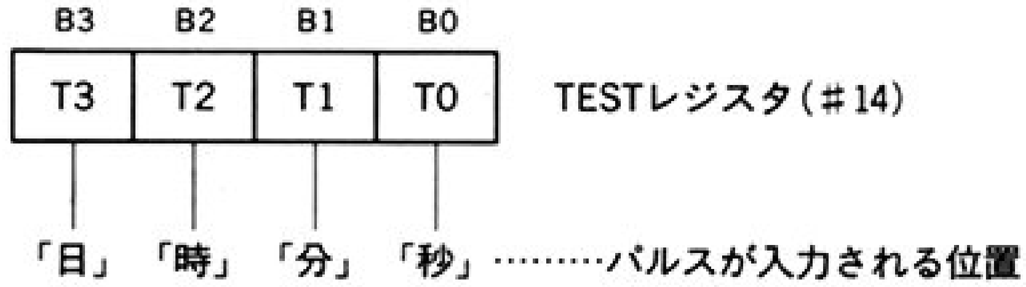


図5.27　TESTレジスタの機能

6.5　RESETレジスタの機能

RESETレジスタ（#15）には以下の機能があります．

● アラームのリセット

ビット0を“1”にすると，すべてのアラームレジスタを0にリセットします．

● 秒合わせ

ビット1を“1”にすると，秒以前の分周段をリセットします．この機能は1秒の始まりを正確に合わせる場合に用います．

● クロックパルスのON／OFF

ビット2を“0”にすると，16Hzのクロックパルス出力をONに，ビット3を“0”にすると，1Hzのクロックパルス出力をONにします．なお，両者ともMSX2標準仕様ではサポートしていません．

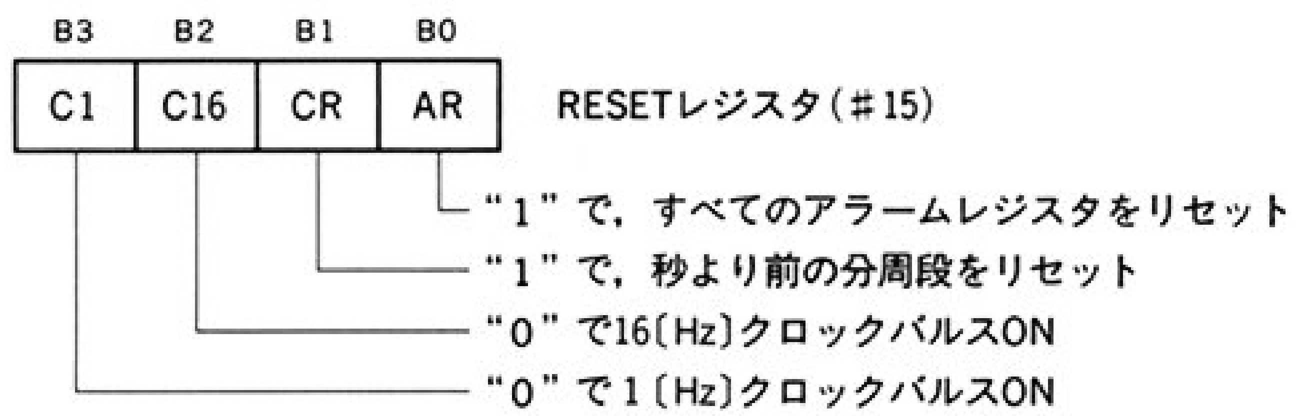


図5.28　RESETレジスタの機能

6.6　クロックおよびアラームの設定

● 日付と時刻の設定

クロックの設定にはブロック0を使用します．MODEレジスタでブロック0を選択し，目的とするレジスタにデータを書き込めば日付や時刻が設定されます．また，そのレジスタの内容を読み出せば現在の時間を知ることができます．レジスタの意味とそのアドレスについては図5.29を参照してください．

アラームの設定はブロック1を使用します．ただしアラーム時刻は日，時，分の単位しか指定できません．また，クロックがアラーム時刻に一致しても一般には何も起こりません．

クロックの中で，年は2桁（レジスタ#11～#12）で表されます．MSX-BASICでは，この値にオフセット80を加えて西暦年数の下2桁を表すことにしています．たとえばレジスタ#11＝0，レ

ジスタ #12＝0，と設定した後 BASIC の GET DATE 命令を用いて日付を読み出すと，"80/XX/XX" のように，年数は 80となっているはずです．

曜日は 0～6 で表されます．これは単に日付とともに更新される 7 進カウンタにすぎず，実際の曜日と 0～6 の数値の対応は決まっていません．

ブロック０：CLOCK

		B3	B2	B1	B0
0	秒(１の位)	×	×	×	×
1	秒(10の位)	・	×	×	×
2	分(１の位)	×	×	×	×
3	分(10の位)	・	×	×	×
4	時(１の位)	×	×	×	×
5	時(10の位)	・	・	×	×
6	曜日	・	×	×	×
7	日(１の位)	×	×	×	×
8	日(10の位)	・	・	×	×
9	月(１の位)	×	×	×	×
10	月(10の位)	・	・	・	×
11	年(１の位)	×	×	×	×
12	年(10の位)	×	×	×	×

ブロック１：ALARM

		B3	B2	B1	B0
0	――	・	・	・	・
1	――	・	・	・	・
2	分(１の位)	×	×	×	×
3	分(10の位)	・	×	×	×
4	時(１の位)	×	×	×	×
5	時(10の位)	・	・	×	×
6	曜日	・	×	×	×
7	日(１の位)	×	×	×	×
8	日(10の位)	・	・	×	×
9	――	・	・	・	・
10	12時 OR 24時	・	・	・	×
11	閏年カウンタ	・	・	×	×
12	――	・	・	・	・

図中に"・"で示したビットは常に０であり，変更できない

図5.29　CLOCKとALARMの設定

● 12時間計／24時間計の選択

時刻の表示は，昼の１時を「13時」と表す24時間計と，「午後１時」と表す12時間計のどちらかを選ぶことができます．この選択にはブロック１のレジスタ #10 を使用します．図５.30 に示したように，B0 が "０" の時12時間計，"１" の時24時間計となります．

12時間計を選んだ場合は，図５.31 のように10時間カウンタ（ブロック０，#5）の B1 ビットによって午前／午後を表現します．

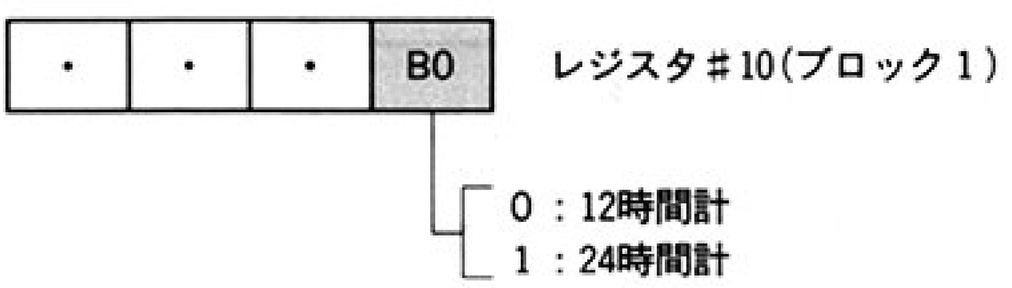


図5.30　12時間計／24時間計の選択

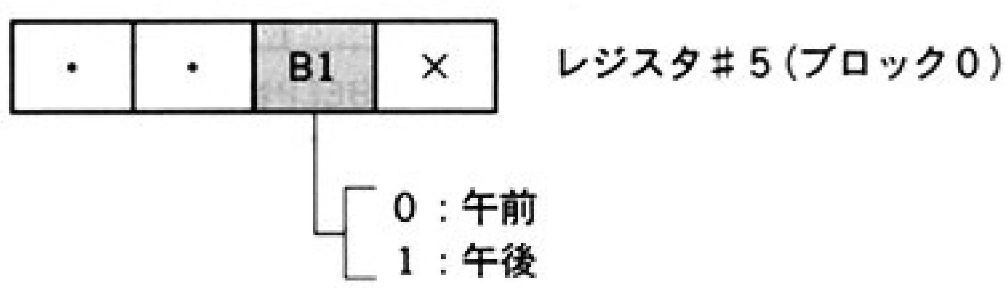


図5.31　12時間計の午前／午後フラグ

● 閏年カウンタ

ブロック1のレジスタ #11は，年のカウントとともに更新される4進カウンタです．このレジスタの下位2ビットが 00H の場合は閏年とみなされ，2月を29日までカウントします．

MSX-BASIC で“SET DATE”命令を実行すると，与えた年を4で割った余りがこのレジスタに設定されます．たとえば“80/XX/XX”という日付を指定すると閏年カウンタは0になりますが，これは西暦1980年が閏年であるという事実に一致するわけです．

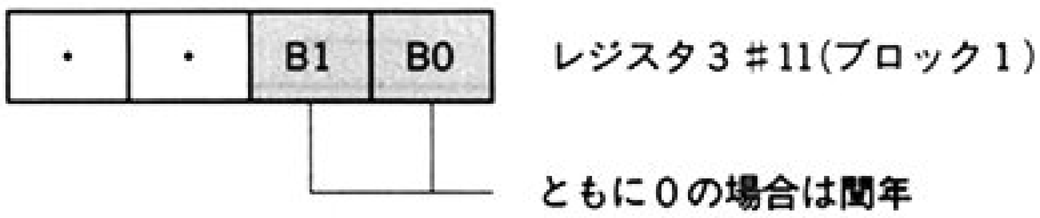


図5.32　閏年の判定

6.7　バッテリバックアップ・メモリの内容

CLOCK-IC のブロック2とブロック3は，それぞれ4ビット×13のバッテリバックアップされたメモリブロックとして用いられ，MSX 2ではこの部分を以下のような用途に使用しています．

● ブロック2の内容

	B3	B2	B1	B0
0	ID			
1	Adjust X(−8〜+7)			
2	Adjust Y(−8〜+7)			
3	——	——	Interlace Mode	Screen Mode
4	WIDTHの値(Lo)			
5	WIDTHの値(Hi)			
6	前景色			
7	背景色			
8	周辺色			
9	Cassette Speed	Printer Mode	Key Click	Key ON/OFF
10	BEEP音色		BEEP音量	
11	——	——	タイトルカラー	
12	国別コード			

図5.33　ブロック2の内容

● ブロック3の内容

ブロック3は，ID値（レジスタ#0）の内容により，3通りの機能を持っています．図5.34にその機能を示します．

ID＝0：初期画面にタイトル（6文字以内）を表示する

0	0	
1 2	Lo1 ─┐─タイトルの1番目の文字 Hi 1 ─┘	6文字
⋮	⋮	
11 12	Lo6 ─┐─タイトルの6番目の文字 Hi 6 ─┘	

ID＝1：パスワードの設定

0	1
1 2 3	Usage ID＝1 Usage ID＝2 Usage ID＝3
4 5 6 7	パスワード パスワード パスワード パスワード　─ パスワードデータを4ビット×4に圧縮して格納する
8	Keyカートリッジ存在フラグ
9 10 11 12	Keyカートリッジの値 Keyカートリッジの値 Keyカートリッジの値 Keyカートリッジの値

ID＝2：BASICのプロンプト設定

0	2	
1 2	Lo1 ─┐─プロンプトの1番目の文字 Hi 1 ─┘	6文字
⋮	⋮	
11 12	Lo6 ─┐─プロンプトの6番目の文字 Hi 6 ─┘	

図5.34　ブロック3の内容

6.8　CLOCK-IC のアクセス

クロックおよびバッテリバックアップ・メモリをアクセスするために，下記のような BIOS ルーチンが用意されています．このルーチンは SUB-ROM に存在するため，一般にはインタースロットコールを用いて呼び出します．

● REDCLK（01F5H ／ SUB） ……………CLOCK-IC のデータの読み出し

入力：C←CLOCK-IC のアドレス（**図5.35**参照）

出力：A←読み出したデータ（下位4ビットのみ有効）

機能：C レジスタで指定したアドレスの CLOCK-IC レジスタを読み出し，A レジスタに格納する．アドレス指定には図5.35のようにブロック選択情報も含めているため，MODE レジスタを設定し，次に目的のレジスタを読み出す，という手間をかける必要はない．

Cレジスタ	・	・	M1	M0	A3	A2	A1	A0
			選択するブロック		レジスタのアドレス			

図5.35　CLOCK-ICのレジスタ指定法

● WRTCLK（01F9H ／ SUB） ……………CLOCK -IC のデータの書き込み

入力：C← CLOCK-IC のアドレス（図5.35参照）

A←書き込むデータ（下位4ビットのみ有効）

出力：——

機能：C レジスタで指定したアドレスの CLOCK-IC レジスタに，A レジスタの内容を書き込む．アドレスは REDCLK と同様に図5.35のようなフォーマットで指定する．

この BIOS ルーチンの使用例を**リスト5.10**に示します．

リスト5.10 プロンプトの設定

```
;=============================================================
;
;   List 5.10  set prompt message
;
;=============================================================
;
WRTCLK  EQU     01F9H
EXTROM  EQU     015FH

        ORG     0B000H

;----- program start -----        BASICのプロンプトメッセージを設定する

START:  LD      C,00110000B       ;address data
        LD      A,2               ;ID := prompt mode
        CALL    WRTRAM            ;write to back-up RAM

        LD      B,6               ;loop counter
        LD      HL,STRING         ;prompt data
L01:    LD      A,(HL)            ;read string data
        AND     0FH               ;A := hi 4 bit
        INC     C                 ;address inc
        CALL    WRTRAM            ;write data to buck-up RAM
        LD      A,(HL)
        RRCA
        RRCA
        RRCA
        RRCA
        AND     0FH
        INC     C                 ;address inc
        CALL    WRTRAM            ;write low 4 bit
        INC     HL
        DJNZ    L01
        RET

;----- write data to buck-up RAM -----

WRTRAM: PUSH    HL
        PUSH    BC
        LD      IX,WRTCLK
        CALL    EXTROM            ;use interslot call
        POP     BC
        POP     HL
        RET

;----- string data -----

STRING: DB      'Ready?'

        END
```

7章 スロットとカートリッジ

MSXで使用されているCPU(Z80)は，通常64Kバイト(0000H～FFFFH)のアドレス空間しかアクセスできません．しかし，MSXは1M(メガ)バイトに相当する空間を自由にアクセスすることができます．これはMSXが“スロット”の概念を採用し，同一のアドレスに複数のメモリあるいはデバイスを割り当てる機能を備えているからです．

本章では，このスロットの使用法，およびスロットを介してMSXにカートリッジソフトや新しいデバイスを接続するために必要となる情報を紹介していきます．

7.1 スロット

スロットは大量のアドレス空間を確保するためのインターフェイスの役目をするもので，MSXのアドレスバスに接続されるメモリやデバイスは，すべてスロットを介して実装されています．それは，本体内にあるBASICのROMであろうとMSX-DOSモード時のRAMであろうと例外ではありません．カートリッジソフトを差す場所も，1つのスロットです．ここでは，スロットに接続されたソフトウェアやデバイスの取り扱い方を説明します．

7.1.1 基本スロットと拡張スロット

スロットには基本スロットと拡張スロットの2種類があります．“基本スロット”とは，図5.36に示すようにCPUのアドレスバスに直結するスロットを指し，MSXの仕様では最大4個持つことができます．基本スロットは，スロット拡張ボックスを接続することにより(本体内で拡張されていることもある)，最高4個のスロットに拡張でき，この時のスロットを“拡張スロット”と呼びます．4個の基本スロットをそれぞれ4つの拡張スロットに拡張した場合，スロットの数は最大の16個となり，16スロット×64Kバイト＝1Mバイトのアドレス空間をアクセスすることが可能です．

なお，拡張スロットにさらに拡張ボックスを差し込んだ場合はシステム自体が起動できなくなりますので，このようなことは行わないでください(MSX標準のカートリッジ用スロットはかならず基本スロットだが，各機種のオプションハードウェア専用コネクタは，拡張スロットに接続されていることがある)．

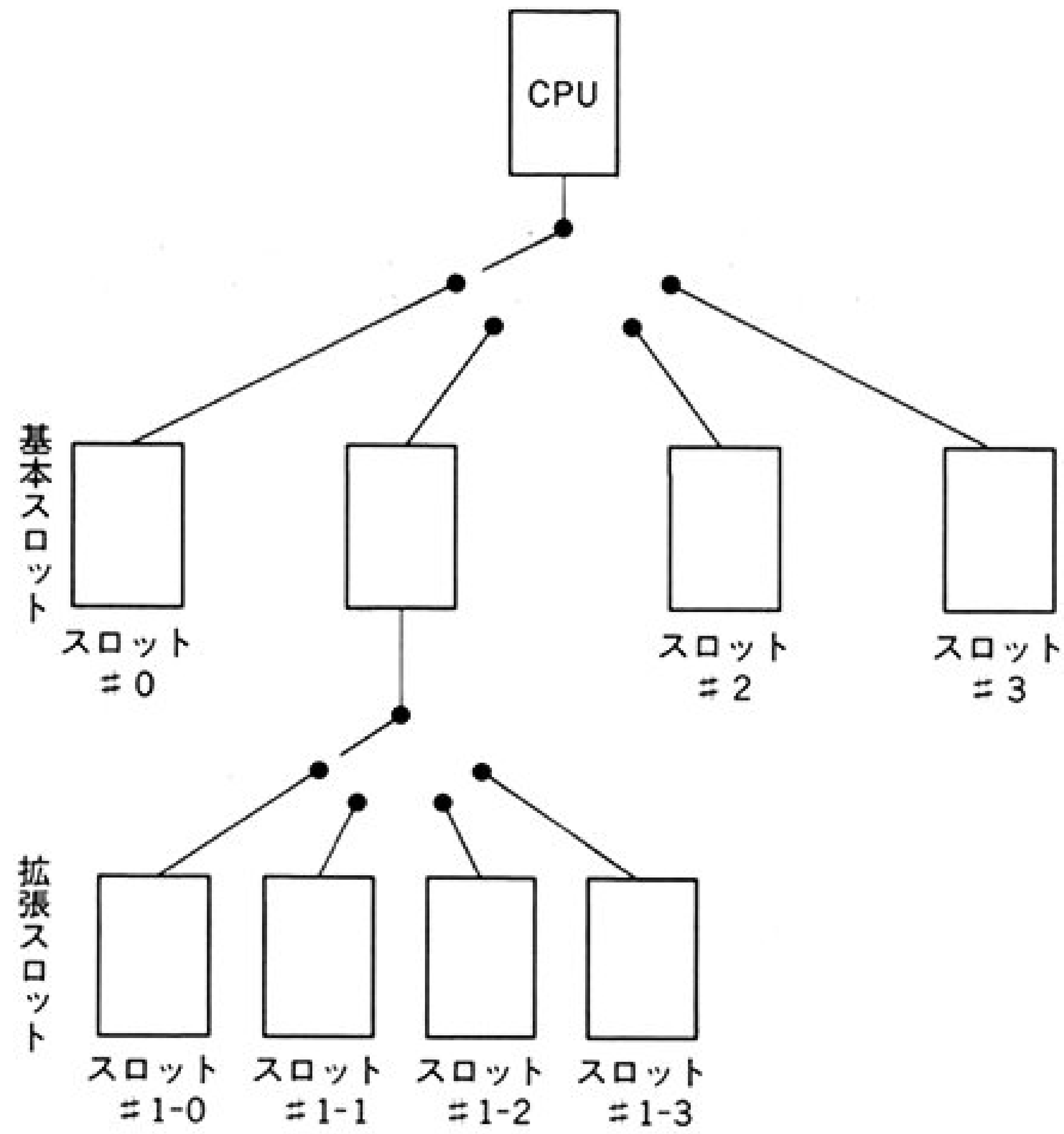


図5.36　基本スロットと拡張スロット

各スロットは0000H～FFFFHまでの64Kバイトのアドレス空間を持ちますが，MSXではそれを16Kバイトずつ4つに分け“ページ”として管理しています．CPUはページごとに任意のスロットを選択してアクセスでき，図5.37のように，いくつかのスロットから必要な部分だけを選んで組み合わせることも可能です．ただし，ある番号のページを異なる番号のページに割り当てることはできません(つまり，各スロットのページnは，CPUから見てもページnである)．

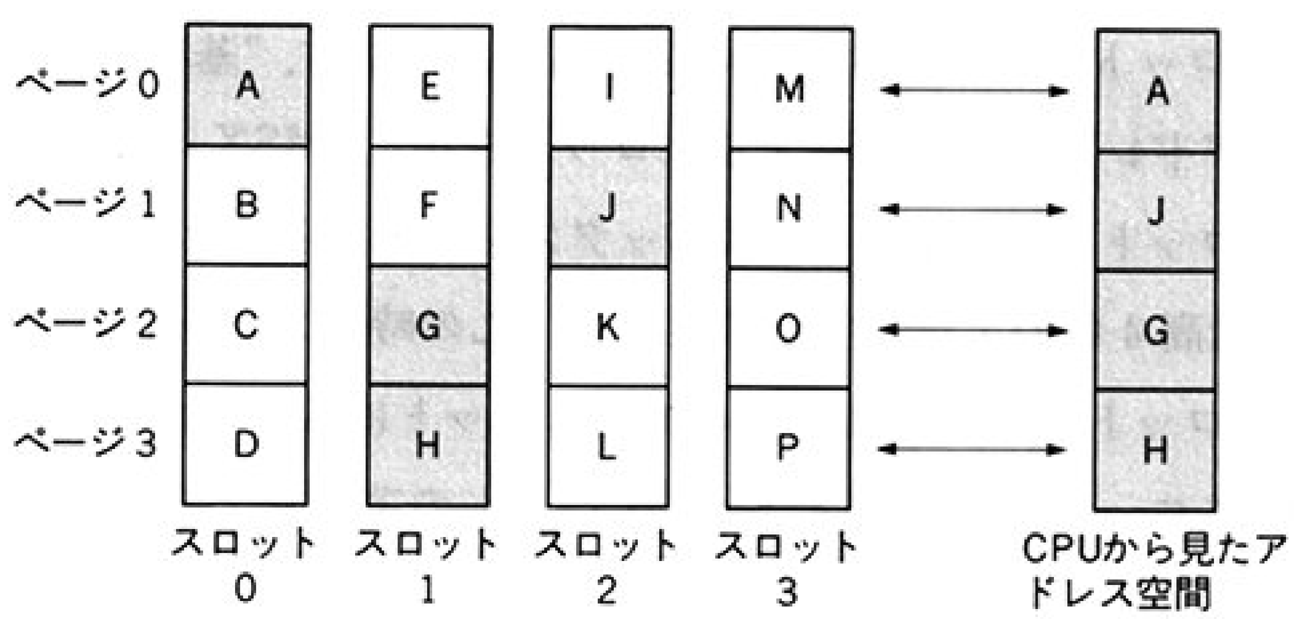


図5.37　ページ選択の例

7.1.2 スロットの選択

スロットの選択方法は，基本スロットと拡張スロットでは異なります．基本スロットの場合はA8H番地のI/Oポートによって行い(図5.38)，拡張スロットの場合は実装された拡張カートリッジの“拡張スロット選択レジスタ(FFFFH)”によって行います(図5.39)．しかし，それらを直接変更することはたいへん危険ですから，スロットの切り換えは不要意に行わないでください．特に，プログラムが自分自身のいるページのスロットを切り換えた場合，動作は保証されません．他のスロットにあるプログラムを呼び出したい場合は，次節で説明するインタースロットコールを使用してください．

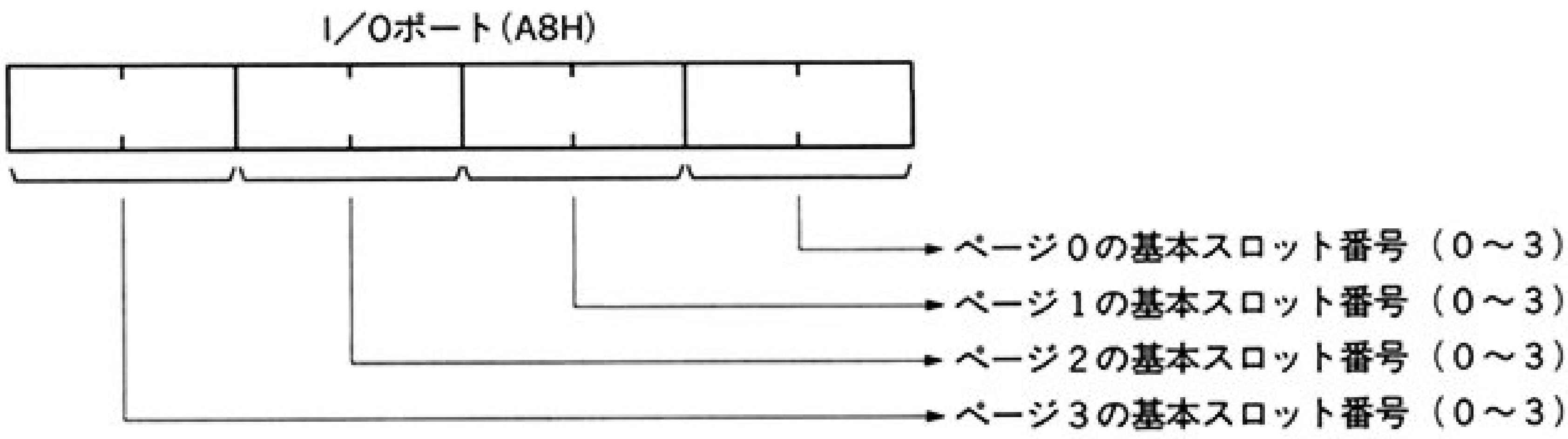
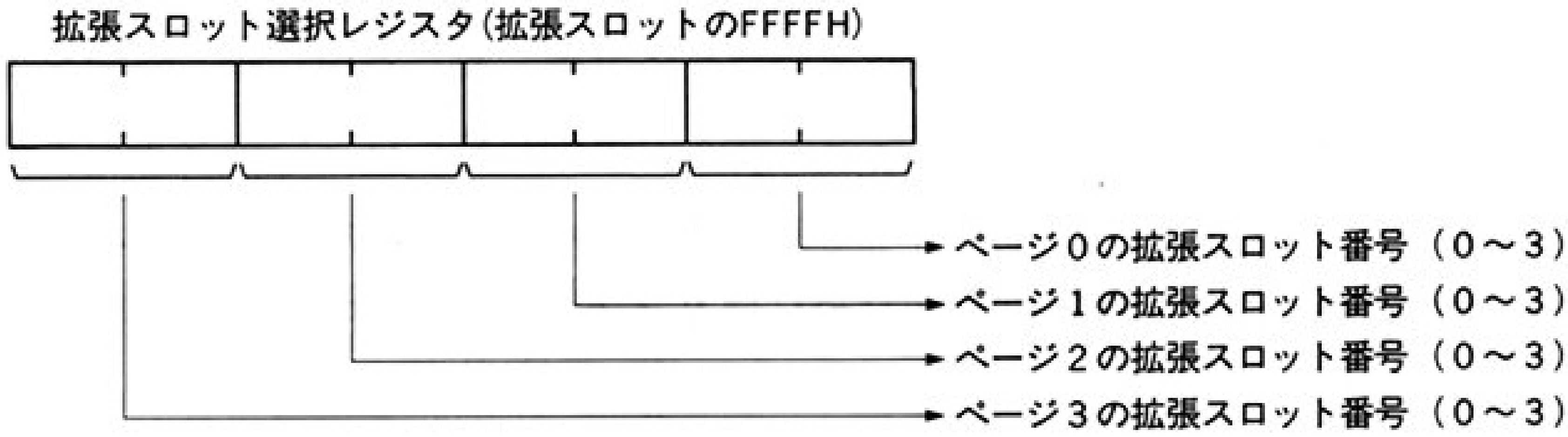


図5.38　基本スロットの選択



注）　このスロットが基本スロットか，拡張スロットかが判るように，拡張スロットの場合は値を書き込んだ後に読み出すと，書き込んだ値の反転した値となって読み出されます．
　このレジスタは同一基本スロット内であれば，どの拡張スロットでも同じ値です．

図5.39　拡張スロットの選択

どこのスロットにMAIN-ROMやRAMが実装されているか，またカートリッジ用のスロットが何番のスロットであるかは機種によって異なります．もし手持ちのMSXがどのようにスロットを使用しているかを知りたい人はマニュアルなどで調べてください．しかし，MSXはどのスロットに何があろうと正常な動作ができるように仕様が決められていますから，その仕様に準じている限り，通常はスロットの使用状況を気にする必要はありません．

しかし，場合によっては特定のソフトウェアが何番のスロット上に置かれているのかを知る必要が生じることもあります．たとえば，従来はBASICのMAIN-ROMは基本スロット#0，または基本スロット#0を拡張した拡張スロット#0-0に置くという仕様でしたが，MSX1にMSX-VIDEOとBASIC ver 2.0のROMを増設してMSX 2の機能を持たせる場合には，MAIN-

ROMがスロット#0やスロット#0-0以外の上に置かれることになります．また，MSX 2のSUB-ROMが入っているスロットは機種によってまちまちで，このような場合，下記のワークエリアを参照することにより，BASICインタープリタのROMが置かれているスロットを知ることが可能です(スロット情報は図5.40のフォーマットで得られる)．DOSからBIOSを呼び出す時も，この方法を使用してMAIN-ROMのスロットを調べてください．

- EXPTBL(FCC1H，1)　MAIN-ROMのスロット
- EXBRSA(FAF8H，1)　SUB-ROMのスロット(MSX 1では0)

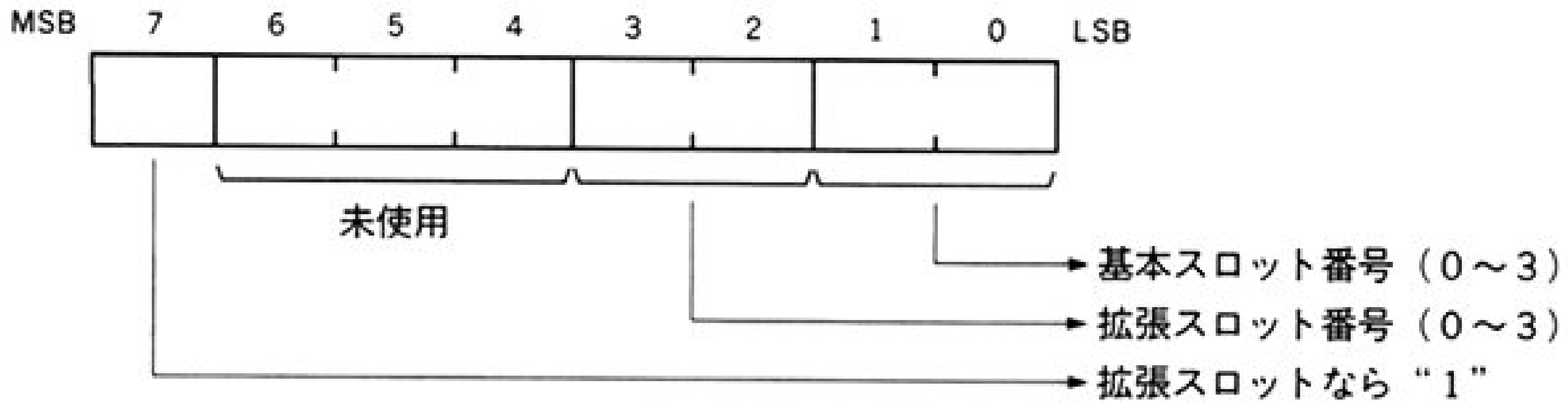


図5.40　スロットのフォーマット

また，あるルーチンがページ1とページ2(4000H～BFFFH)にまたがっている場合を考えてください．このルーチンの中で1ページから2ページへ分岐が起こる時には，ページ2にページ1と同じスロットが選択されていなければなりません．そのためには，ページ1でまず自分のいるスロットを調べ，ページ2をそのスロットに切り換えるという作業が必要です．現在自分がどのスロットにいるのか，という情報を知るためには**リスト5.11**のプログラムを実行してください．

リスト5.11　自分のいるスロットを知るためのプログラム

```
;=============================================================
;  List 5.11    to know where you are
;=============================================================
;       Suppose your  program  cartridge  is  32K  bytes
;       long (4000H..0BFFFH).   You  set the ID at 4000H
;       and 4001H and the execution start address within
;       page 1   (4000H..7FFFH).     MSX  passes  control
;       to this address so the  part  which  resides  in
;       page 2  is  not  yet enabled at this point.  You
;       have to know where  you  are  (in  what  primary
;       slot, in  what  secondary  slot)  and enable the
;       part at page 2.  Below  is  the  sample  program
;       to do this.
;
ENASLT   EQU      0024H              ;enable slot
RSLREG   EQU      0138H              ;read primary slot select register
EXPTBL   EQU      0FCC1H             ;slot is expanded or not

;----- program start -----
```

```
ENAP2:
        CALL    RSLREG          ;read primary slot #
        RRCA                    ;move it to bit 0,1 of [Acc]
        RRCA
        AND     00000011B
        LD      C,A
        LD      B,0
        LD      HL,EXPTBL       ;see if this slot is expanded or not
        ADD     HL,BC
        LD      C,A             ;save primary slot #
        LD      A,(HL)          ;get the slot is expanded or not
        AND     80H
        OR      C               ;set MSB if so
        LD      C,A             ;save it to [C]
        INC     HL              ;Point to SLTTBL entry
        INC     HL
        INC     HL
        INC     HL
        LD      A,(HL)          ;Get what  is  currently  output
                                ;to expansion   slot    register
        AND     00001100B
        OR      C               ;Finaly form slot address
        LD      H,80H
        JP      ENASLT          ;enable page 2
 :
        END
```

7.2 インタースロットコール(スロット間コール)

前述のように，MSX ではプログラムが異なったスロットに分かれているため，現在選択されているスロット上にないプログラムが必要になることもあります．これは主に以下のような場合が考えられます．

(1)MSX-DOS レベルから，MAIN-ROM にある BIOS を呼び出す．
(2)BASIC レベルから，SUB-ROM にある BIOS を呼び出す(MSX 2 のみ)．
(3)カートリッジソフトから，MAIN-ROM あるいは SUB-ROM の BIOS を呼び出す．

これらの作業を行う際スロット切り換えが簡単かつ安全に行えるように，インタースロットコールという一群の BIOS ルーチンが存在し，どのスロットに存在するルーチンでも呼び出せるようになっています．本節では，このインタースロットコールの使用法を説明しましょう．

7.2.1 インタースロットコールの動作

たとえば MSX-DOS から MAIN-ROM 上の BIOS を呼び出す場合，スロットの状態の遷移は以下に示すとおりです．

(1)初めはMSX-DOSモードなので64Kのアドレス空間すべてにRAMが選択されており，このままではBASIC-ROMをアクセスできない(図5.41-イ).

(2)ROMのBIOSを呼び出すためにページ0をBASICのMAIN-ROMに切り換え，アクセス可能な状態にする．そしてBIOSを呼び出す(図5.41-ロ).

(3)BIOSの処理が終わったら再び元の状態に戻し，初めに呼ばれたアドレスにリターンする.

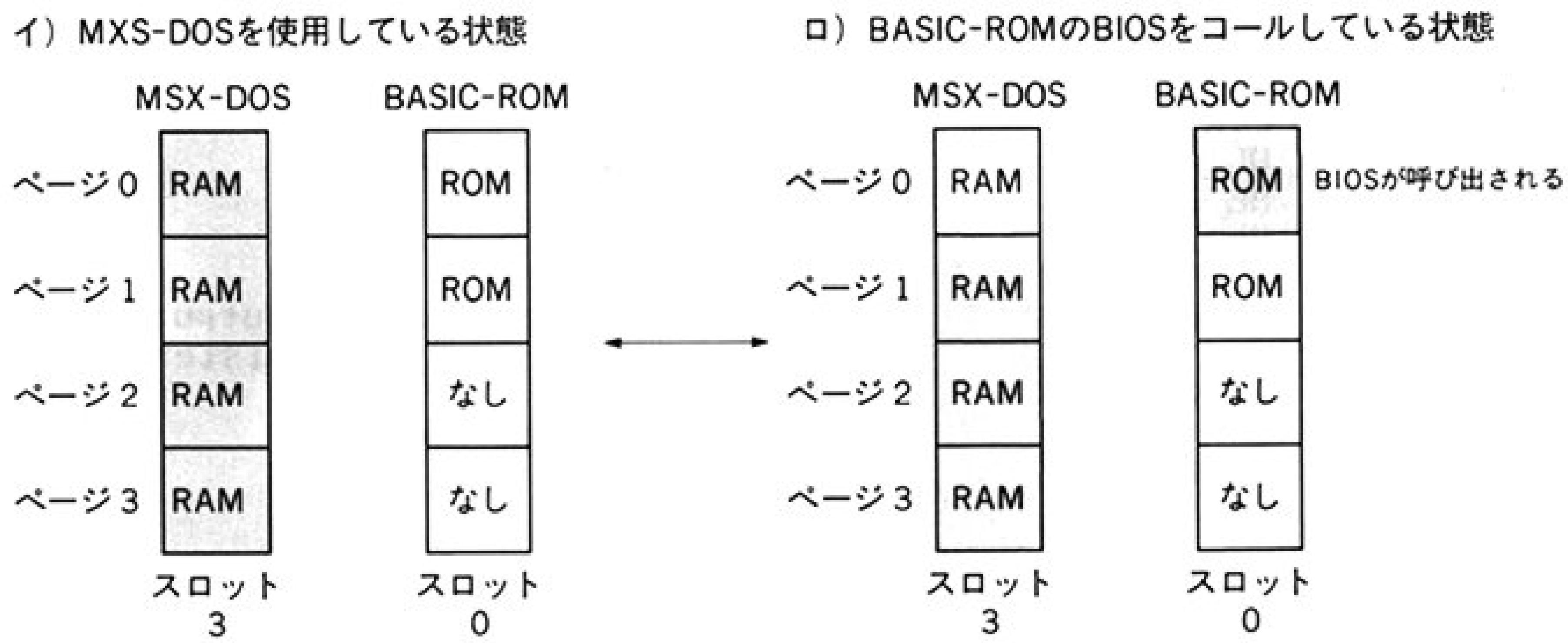


図5.41　インタースロットコールのされ方

この時，MSX-DOS側のプログラムが，ページ0以外に存在するならば話は簡単なのですが，呼び出す側のプログラムが呼び出される側のBIOSと同じページ0に存在している時は，そう単純にはいきません．呼び出す方のプログラムがページ0を切り換えたとたんに自分自身がいなくなって暴走してしまう，ということのないように配慮しなければならないのです．インタースロットコールでは，いったんページ3へ分岐してから実際のスロット切り換えを行うという方法で，この問題を解決しています.

7.2.2 インタースロットコールの使用法

インタースロットコールは，以下に述べるいくつかの方法で行うことができます．これらはBIOSとしてMAIN-ROM内に含まれているものですが，その中のいくつかはMSX-DOSの環境下においてもまったく同一のものが用意され，MSX-DOS使用時のインタースロットコールを可能としています.

(1) BIOSのインタースロットコール・ルーチン

● RDSLT(000CH／MAIN) ……… 指定スロットの指定アドレスから値を読む

入力：Aレジスタ←スロット指定

HLレジスタ←読み出すアドレス

出力：Aレジスタ←読み出した値

使用：AF, BC, DE

機能：指定したスロットの，指定したアドレスの内容を読み出し，Aレジスタに格納する．スロットの指定はAレジスタによって図5.40の形式で行う．この時，目的のスロットが基本スロットならば，上位6ビットはすべて“0”に設定し，下位2ビットでスロット#0～#3を決める．もし拡張スロットを指定するならば，同様にビット0とビット1で基本スロットを指定し，その基本スロットに接続されるどの拡張スロットかということをビット2とビット3で指定，さらにビット7を“1”とする．

● WRSLT(0014H／MAIN)　………　指定スロットの指定アドレスに値を書き込む

入力：Aレジスタ←スロット指定(図5.40と同じフォーマット)
　HLレジスタ←書き込むアドレス
　Eレジスタ←書き込む値

出力：———

使用：AF, BC, D

機能：Aレジスタで指定したスロット(指定の形式は図5.40と同じ)の，HLレジスタで指定したアドレスに，Eレジスタの値を書き込む．

● CALSLT(001CH／MAIN)　………　指定したスロットの指定アドレスをコールする

入力：IYレジスタの上位8ビット←スロット(図5.40と同じフォーマット)
　IXレジスタ←コールするアドレス

出力：呼び出し先プログラムの実行結果により異なる

使用：呼び出し先プログラムの実行結果により異なる

機能：IYレジスタの上位8ビットで指定したスロット(指定の形式は図5.40と同じ)の，IXレジスタで指定するアドレスに存在するルーチンをコールする．

● ENASLT(0024H／MAIN)　………　スロットを切り換える

入力：Aレジスタ←スロット(図5.40と同じフォーマット)
　HLレジスタ←上位2ビットでスロット切り換えを行うページを指定する

出力：———

使用：すべて

機能：HLレジスタの上位2ビットで指定したページを，Aレジスタで指定したスロットに切り換える．

● **CALLF(0030H／MAIN)** ……… 指定スロットの指定アドレスをコールする

入力：インライン・パラメータ形式でスロットとアドレスを指定する

出力：呼び出し先のプログラムの実行結果により異なる

使用：呼び出し先のプログラムの実行結果により異なる

機能：指定したスロットの，指定したアドレスをコールするが，前述のCALSLTと異なり，スロットおよびアドレスの指定は次に示すように，インライン・パラメータ形式で行う．すなわち，このCALLFを呼び出す命令の直後にスロットを指定する値1バイト(RDSLTと同じフォーマット)を置き，その次にアドレスを指定する数値2バイトを置くという形でパラメータを渡すのである．“CALL 0030H”の代わりに“RST 30H”というRST(リスタート)命令を使ってもよい．その場合，4バイトでインタースロットコールが実現可能である．

```
RST      30H              ;interslot call
DB       00000000B        ;select slot#0
DW       006CH            ;call address = 006CH
```

図5.42　インタースロットコールの実行例

● **RSLREG(0138H／MAIN)** ……… 基本スロット選択レジスタの読み出し

入力：―――

出力：Aレジスタ←読み込んだ値

使用：―――

機能：基本スロット選択レジスタの内容を読み出し，Aレジスタに入れる．

● **WSLREG(013BH／MAIN)** ……… 基本スロット選択レジスタへの書き込み

入力：Aレジスタ←書き込む値

出力：―――

使用：―――

機能：基本スロット選択レジスタにAレジスタの値を書き込み，スロットを選択する．

● **SUBROM(015CH／MAIN)** ……… SUB-ROMの指定したアドレスをコールする

入力：IXレジスタ←コールするアドレス，PUSH IX (352ページ参照)

出力：呼び出し先のプログラムの実行結果により異なる

使用：裏レジスタとIX，IYレジスタは保存される

機能：特にBASICのSUB-ROMを呼び出すためのルーチンである．SUB-ROMの存在するスロットは自動的に調べられる．普通は，次のEXTROMを使う．

● EXTROM(015FH／MAIN)　………　SUB-ROMの指定したアドレスをコールする

入力：IXレジスタ←コールするアドレス

出力：呼び出し先のプログラムの実行結果により異なる

使用：裏レジスタとIYレジスタは保存される

機能：BASICのSUB-ROMを呼び出すためのルーチンである．上記のSUBROMとこのEXTROMとは，IXレジスタの値をプッシュするか否かという点だけが異なる．

(2)　MSX-DOSのインタースロットコール

MSX-DOSでは，以下に示す5種類のインタースロットコールが用意され，MSX-DOSジャンプベクタにそのエントリアドレスが定義してあります．これらはBIOS内の同名のルーチンとまったく同じものですから，機能や使用法に関しては，上記のBIOSをご覧ください．なお，MSX-DOSからSUB-ROM内のルーチンを呼び出す場合，これらのルーチンを使用してはいけません．

- RDSLT(000CH)　………　指定スロットの指定アドレスから値を読み出す
- WRSLT(0014H)　………　指定スロットの指定アドレスに値を書き込む
- CALSLT(001CH)　………　指定スロットの指定アドレスを呼び出す
- ENASLT(0024H)　………　指定スロットを使用可能な状態にする
- CALLF(0030H)　………　指定スロットの指定アドレスを呼び出す

リスト5.12　MSX-DOSからBIOSを呼び出す

```
;==========================================================
;   List 5.12     How to use BIOS from MSX-DOS.
;==========================================================
;
CALSLT   EQU     001CH                      ;Inter slot call
EXBRSA   EQU     0FAF8H                     ;Slot address of BIOS (main) ROM
EXPTBL   EQU     0FCC1H                     ;Slot address of extendex ROM
;
;        LD      IY, (EXPTBL-1)             ;Load slot address of the BIOS ROM
;                                           ;in high byte of IY
;        LD      IX, addres of the BIOS jump table
;        CALL    CALSLT
;
;
;
;
;
;
;
```

```
;----- Sample program to set text mode -----

INITXT  EQU     006CH
LINL40  EQU     0F3AEH
;
TOTEXT: LD      B, 40
        LD      A,(EXBRSA)      ;slot address of SUB-ROM
        OR      A               ;0 if MSX1
        JR      Z,TO40
        LD      B, 80
TO40:   LD      (LINL40),B      ;set width into work area
        LD      IX,INITXT
        LD      IY,(EXPTBL-1)   ;get expanded slot status to IYH
        CALL    CALSLT          ;perform an inter-slot call
        EI                      ;because CALSLT do DI
        RET

        END
```

7.2.3 スロットの状態を知るためのワークエリア

スロットに関するワークエリアには，以下のものがあります．

● **EXBRSA(FAF8H,1)**　………　SUB-ROMのスロット

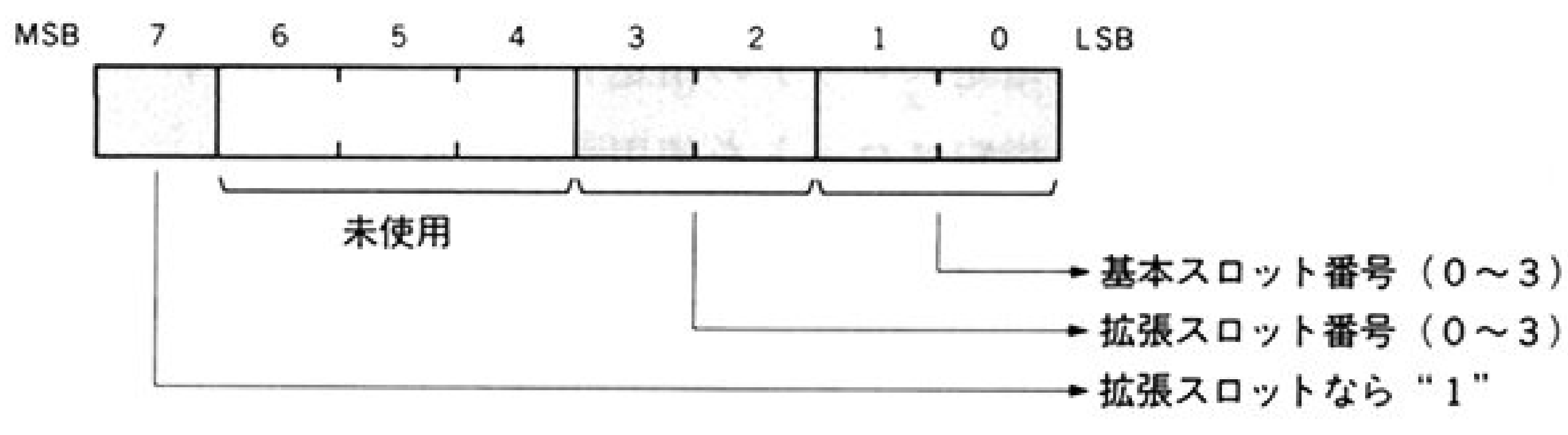


図5.43　SUB-ROMのスロット

● **EXPTBL(FCC1H,4)**　………　基本スロットの拡張の有無

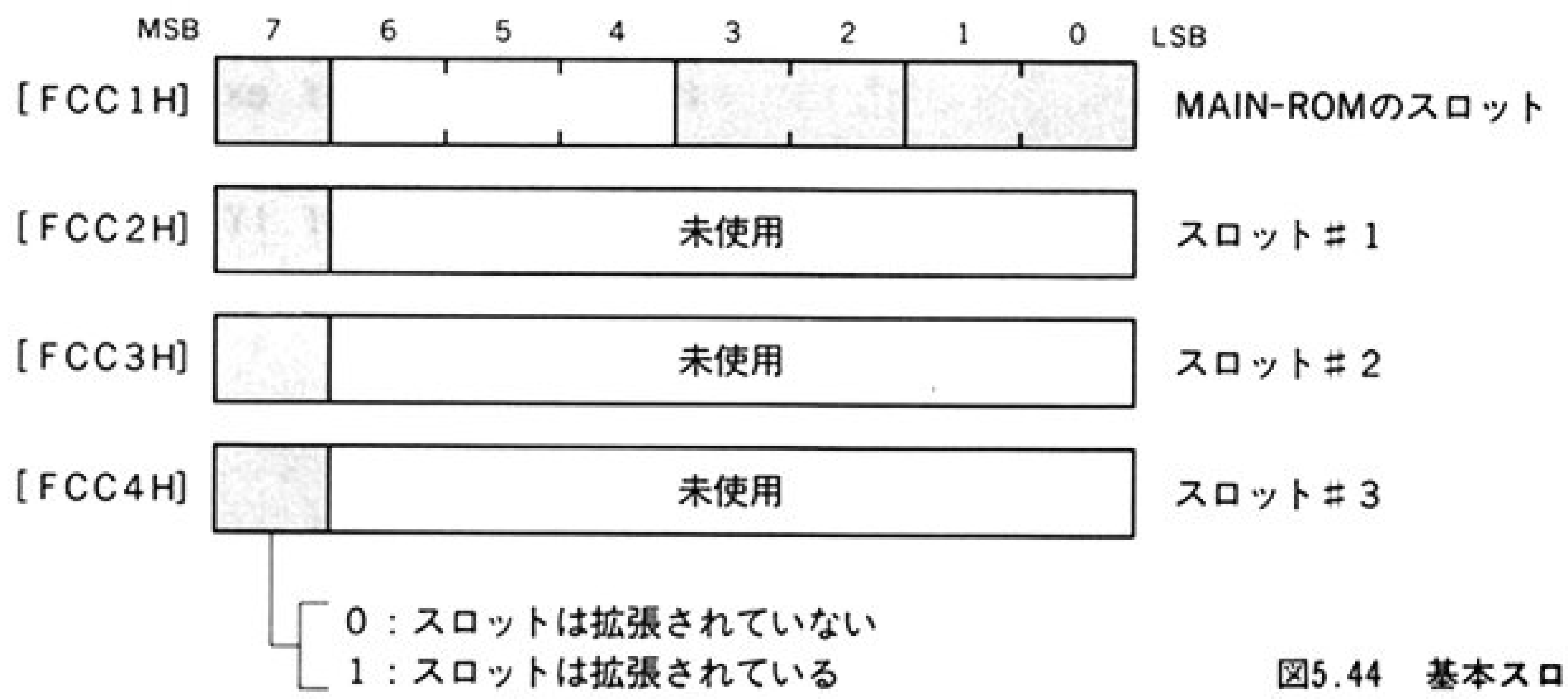


図5.44　基本スロットの選択

● SLTTBL(FCC5H, 4) ……… 拡張スロット選択レジスタ値の保存エリア

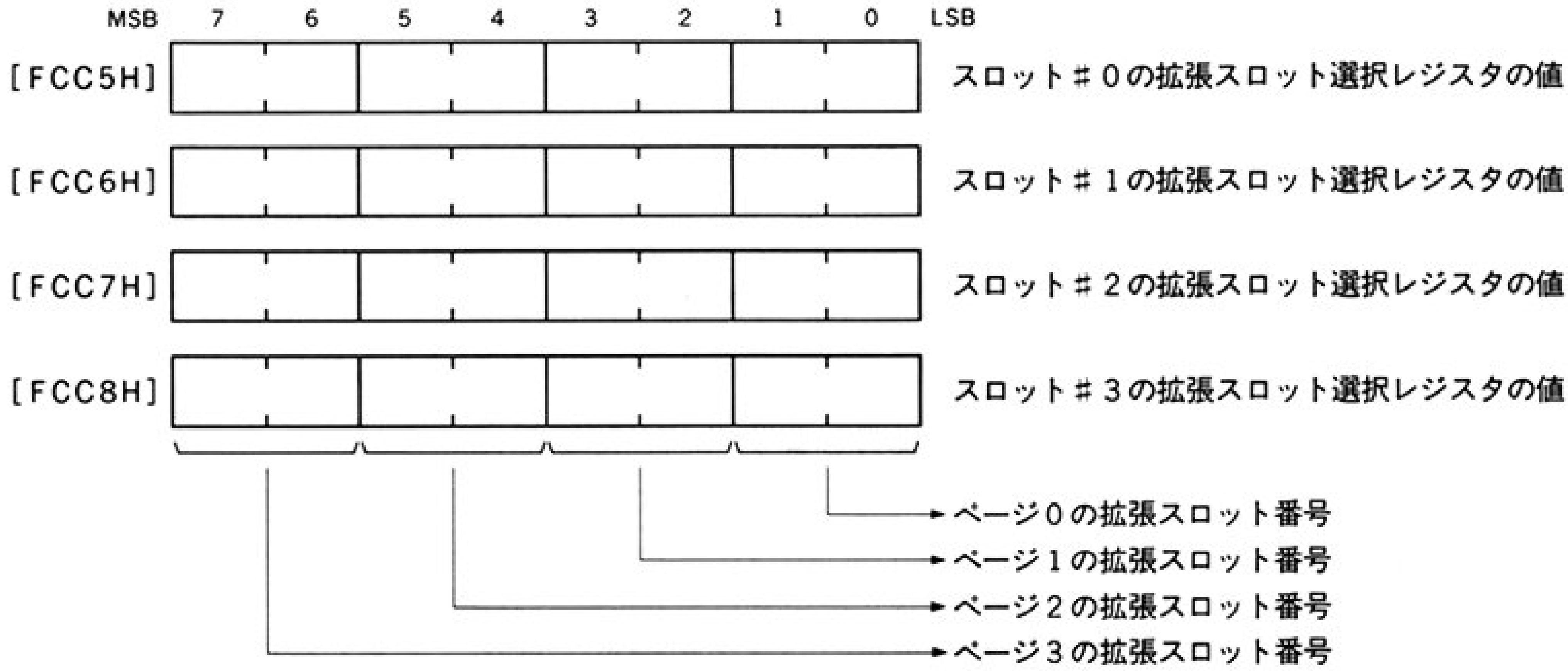


図5.45　拡張スロットの選択

● SLTATR(FCC9H, 64) ……… 各スロット／ページにおけるアプリケーションの有無

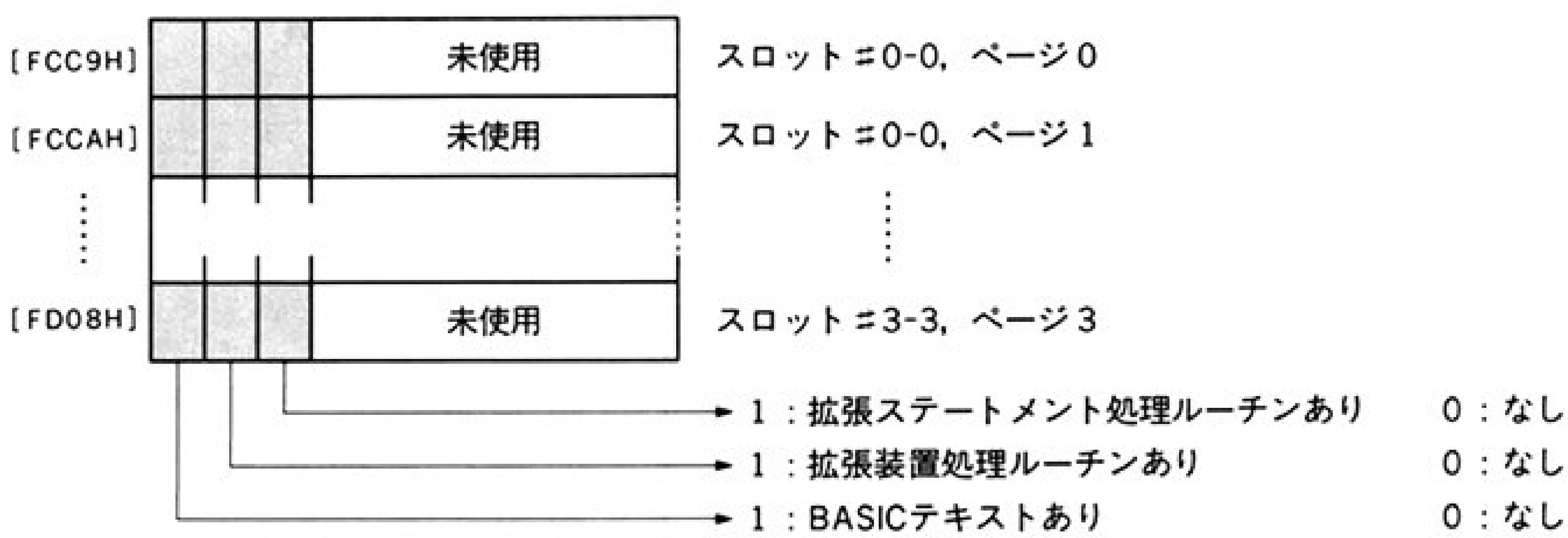


図5.46　アプリケーションの有無

● SLTWRK(FD09H, 128) ……… アプリケーション用ワークエリア

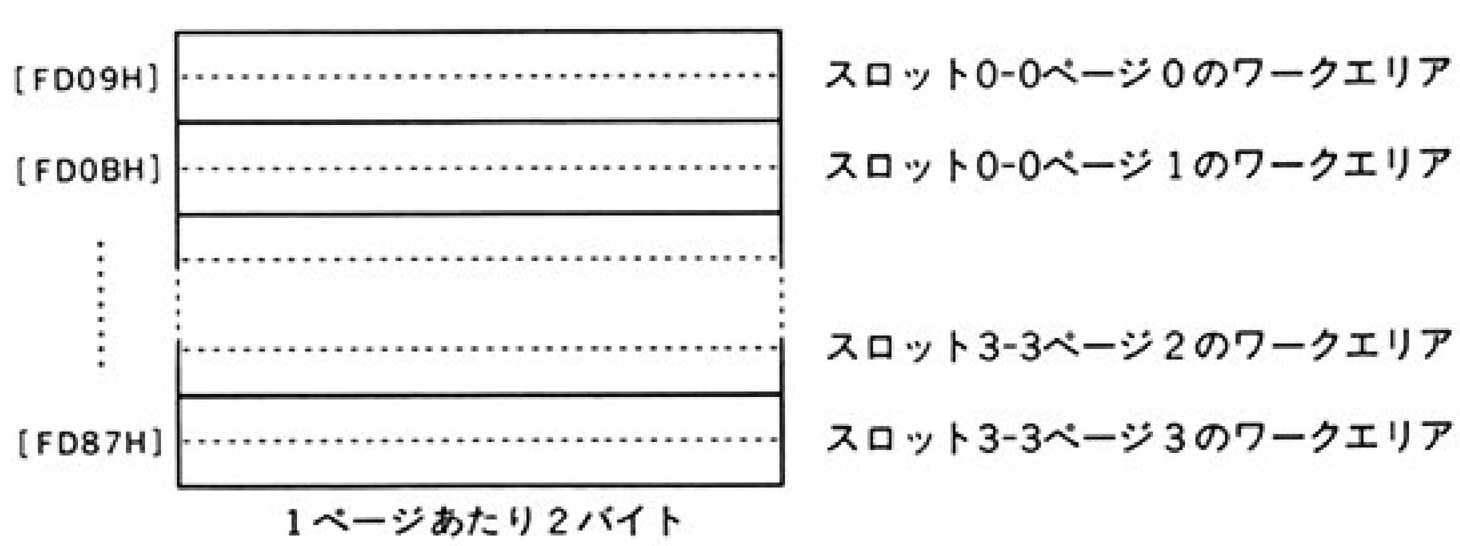


図5.47　アプリケーション用ワークエリア

7.3　カートリッジソフトの作成法

MSXには最低1つのスロットが表に出ており，ここに差し込まれるハードウェアを“カートリッジ”といいます．カートリッジにはアプリケーション・プログラムやゲームなどのROMカートリッジ，ディスク・インターフェースやRS-232Cインターフェイスなどの入出力装置カートリッジ，RAMを増やすための増設RAMカートリッジ，スロットを増やすためのスロット拡張ボックスなどいろいろと接続でき，これによりMSXは機能の拡大を簡単に行うことができます．また，BASICやマシン語プログラムをROM化(カートリッジ化)することも簡単です．ここではソフトウェアをカートリッジ化する方法を説明します．

7.3.1　カートリッジ・ヘッダ

MSXのカートリッジは，16バイトの共通のヘッダを持っており，システムがリセットされると，このヘッダに記述された情報によりカートリッジの初期設定が行われます．BASICやマシン語のプログラムをROM化したカートリッジの場合は，このヘッダに書き込む情報によってオートスタートさせることも可能です．カートリッジ・ヘッダの構成を図5.48に示します．

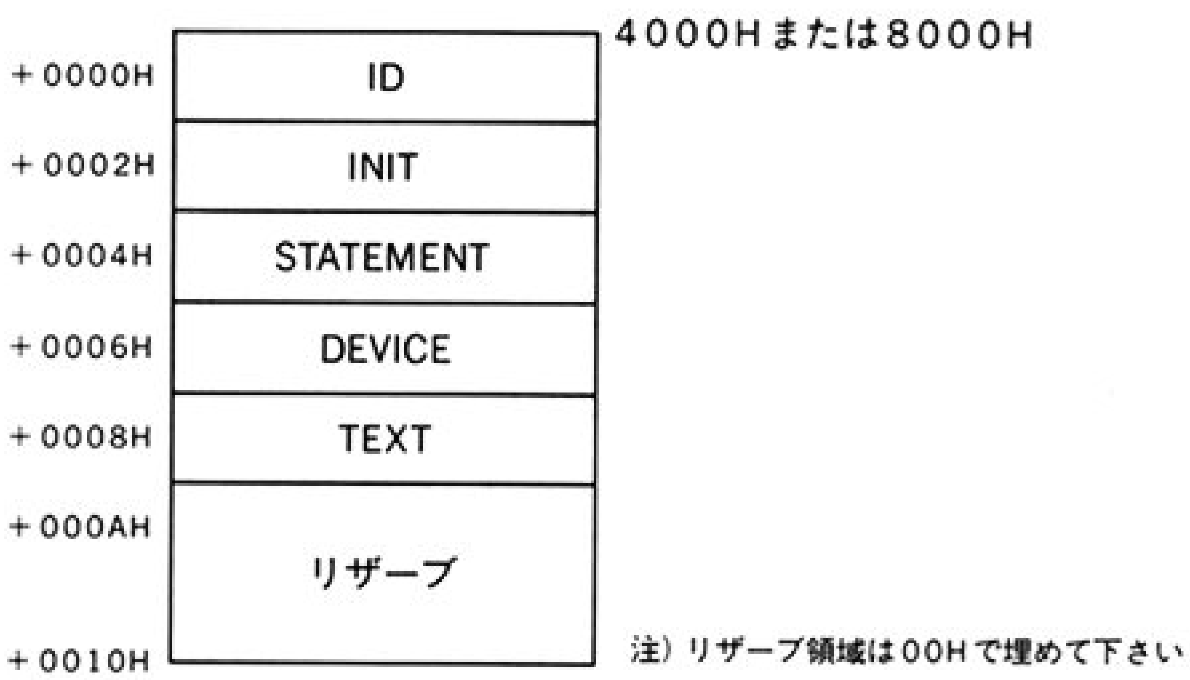


図5.48　プログラムカートリッジのヘッダ

● ID

ROMカートリッジであれば，この2バイトに“AB”(41H，42H)というコードを入れておきます．ちなみにSUB-ROMのカートリッジのIDは“CD”となっています．

● INIT

この2バイトには，カートリッジがワークエリアやI／Oなどの初期化を行うのであれば，その初期化ルーチンのアドレスを書き込み，行わなければ0000Hとしておきます．初期化ルーチンの中でワークエリアの確保など必要な処理を行ったら，“RET”で終了させてください．この時SP

以外のレジスタは内容を破壊してもかまいません．なお，ゲームのようにカートリッジ内でループしていればよいマシン語プログラムの場合は，ここからそのまま目的のプログラムを実行することが可能です．

● STATEMENT

この2バイトには，カートリッジがCALL文の拡張を行うのであれば，ステートメント拡張ルーチンのアドレスを書き込み，行わなければ0000Hとしておきます．CALL文の拡張を行う場合，ステートメント拡張ルーチンは4000H～7FFFHになければいけません．

CALL命令は以下の書式で記述されます．

CALL<拡張ステートメント名> [(<引数> [,<引数>…])]

拡張ステートメント名は15文字以下でなければいけません．CALLの省略形として"_"(アンダースコア)も使用できます．

BASICインタープリタはCALL文を見つけると,ワークエリアのPROCNM(FD89H, 16)に拡張ステートメント名を入れ，ヘッダのSTATEMENTの内容が0以外のカートリッジにスロット番号の小さい方から順に制御を渡してきます．この時HLレジスタは拡張ステートメント名の次のテキストアドレスを指しています(図5.49 a)

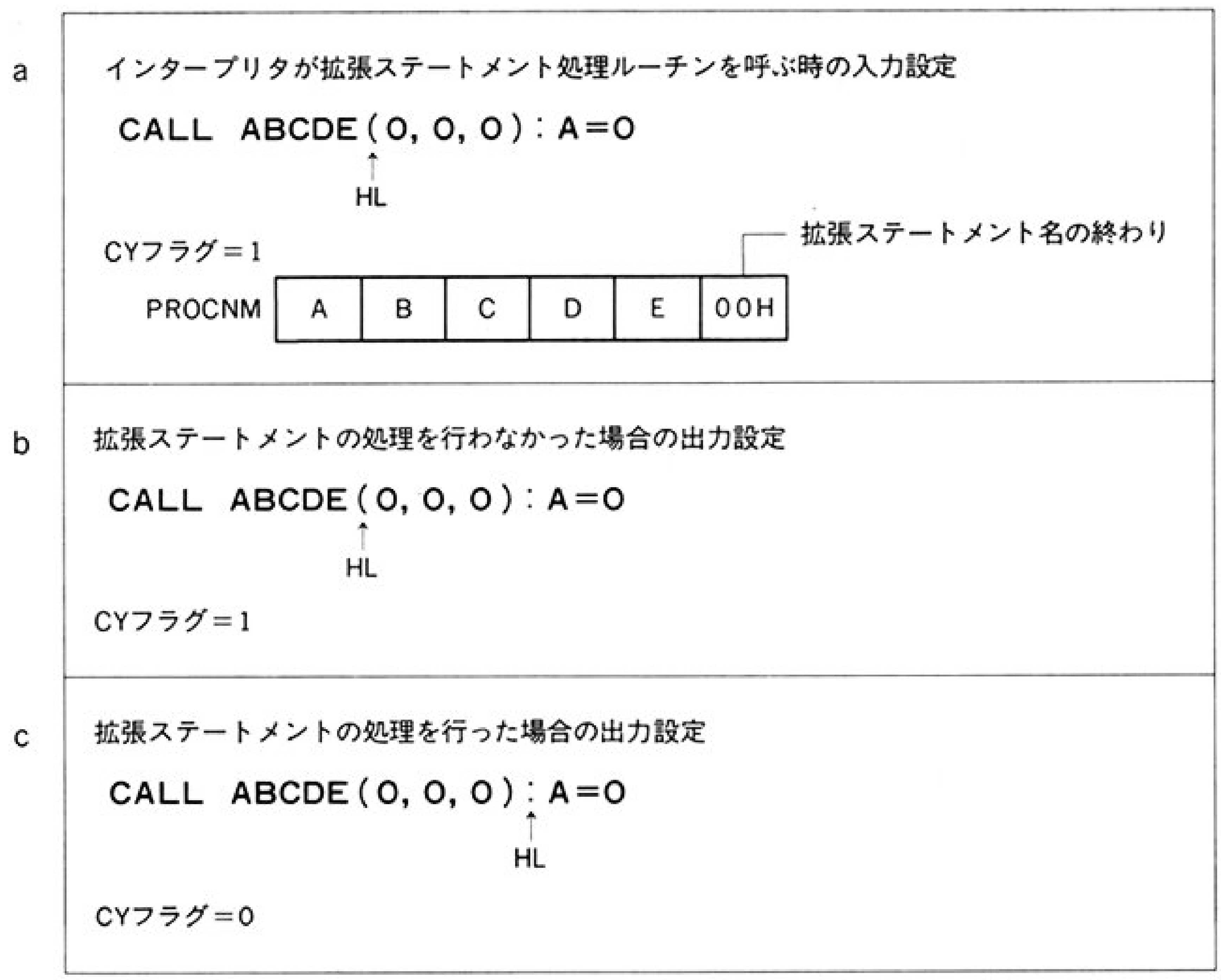


図5.49　拡張ステートメント処理ルーチンの入出力

そこで，ステートメント拡張ルーチンを作成する場合は，まずPROCNMに書かれた拡張ステートメント名を識別し，もしそれが自分の処理すべきものではなかった場合は，HLレジスタの変更はせずに，キャリーフラグを"1"にしてリターンしてください(図5.49b)．拡張ステートメント名が自分の処理すべきものであった場合は，それに応じた処理をし，HLレジスタ(テキストポインタ)を自分が処理したステートメントの次(普通は00Hまたは3AHが入っているところ)に更新し，最後にキャリーフラグを"0"にしてリターンしてください(図5.49c)．

BASICインタープリタはキャリーフラグの状態によって，CALL文が処理されたかどうかを判定し，処理されていない場合は次のカートリッジを呼び出します．すべてのカートリッジが処理をしなかった場合(最後までキャリーフラグが"1"だった時)は"SYNTAX ERROR"を表示します．ステートメントの引数評価には第2部4.4節の"ステートメント拡張のための内部ルーチン"を利用すると便利です．

● DEVICE

この2バイトには，カートリッジがデバイスの拡張(入出力装置の拡張)を行うのであれば，デバイス拡張ルーチンのアドレスを書き込み，行わなければ0000Hとしておきます．デバイスの拡張を行う場合，デバイス拡張ルーチンは4000H～7FFFHになければいけません．デバイスは1つのカートリッジについて，4つまで持つことができます．また，拡張デバイス名は15文字以下でなければいけません．

BASICインタープリタは定義されていないデバイス名を見つけると，PROCNM(FD89H, 16)にそれを格納し，Aレジスタに FFH を入れて，DEVICEの内容が0以外のカートリッジにスロット番号の小さい方から順に制御を渡してきます(図5.50a)．

そこで，デバイス拡張ルーチンを作成する場合は，初めにPROCNMのファイルディスクリプタを識別し，もしそれが自分の処理すべきでないデバイスに対するものだった場合は，キャリーフラグを1にしてリターンしてください(図5.50b)．自分の処理すべきデバイスに対するものだった場合は，それに応じた処理を行い，装置ID(0～3)をAレジスタに設定し，最後にキャリーフラグを0にしてリターンしてください(図5.50c)．

BASICインタープリタはキャリーフラグの状態によって処理されたかどうかを判定し，もし処理されていない場合は次のカートリッジを呼び出します．すべてのカートリッジが処理をしなかった場合(最後までキャリーフラグが"1"だった時)は"Bad file name ERROR"を表示します．

実際の入出力操作をする時には，BASICインタープリタはDEVICE(FD99H)に装置ID(0～3)を，Aレジスタにデバイスに対する要求(表5.6)を設定してデバイス拡張ルーチンを呼び出します．デバイス拡張ルーチンではそれを参照し，要求に応じた処理を行ってください．

a　インタプリタが拡張デバイス処理ルーチンを呼ぶ時の入力設定

OPEN “ABC：” ………

Aレジスタ＝FFH
CYフラグ＝1

ファイルディスクリプタの終わり

PROCNM				
	A	B	C	00H

b　拡張デバイス処理を行わなかった場合の出力設定

CYフラグ＝1

c　拡張デバイス処理を行った場合の出力設定

Aレジスタ＝装置ID（0～3）
CYフラグ＝0

図5.50　デバイス拡張ルーチンへの入出力

Aレジスタ	要　　求
0	OPEN
2	CLOSE
4	ランダムアクセス
6	シーケンシャル出力
8	シーケンシャル入力
10	LOC関数
12	LOF関数
14	EOF関数
16	FPOS関数
18	バックアップキャラクタ

表5.6　デバイスに対する要求

● TEXT

この2バイトには，カートリッジ内のBASICプログラムをオートスタート(リセット時に実行)実行するのであれば，BASICプログラムのテキストポインタを入れておき，行わなければ0000Hとしておきます．プログラムのサイズは8000H～BFFFHの16K以下でなければいけません．

BASICインタープリタはイニシャライズ(INIT)を終え，システムを起動した後，ヘッダのTEXTの内容をスロット番号の小さい順に調べます．そして，そこが0000H以外であった場合，そのアドレスをBASICテキストポインタとしてそこから実行を始めるようになっています(図5.51)．この時のBASICプログラムは，中間コード形式で格納されている必要があり，また

BASIC プログラムの初め(TEXT が指すアドレス)にはプログラムの開始を示す 00H が付いていなければいけません.

なお, GOTO 文などの分岐先の行番号を, あらかじめ分岐先のテキストポインタの絶対アドレスにしておくと, プログラムの実行速度が上がります.

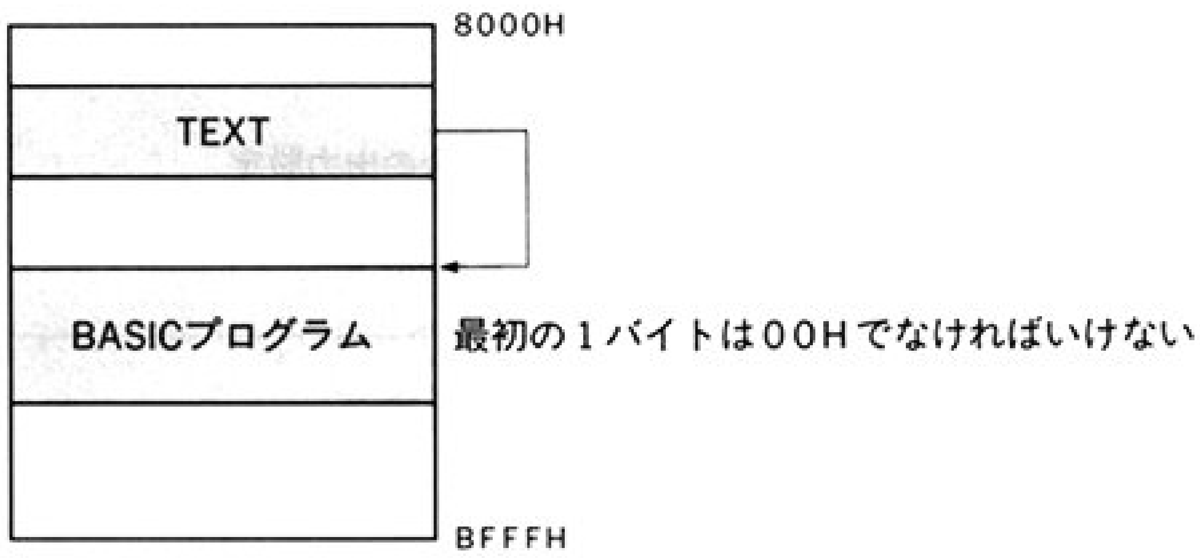


図5.51　BASICプログラムカートリッジの実行

● BASIC プログラムの ROM 化の方法

1 BASIC のテキスト格納先頭アドレスを 8021H に変える.

```
POKE &HF676,&H21 : POKE &HF677,&H80:POKE &H8020,0 : NEW
```

（注）必ず1行にまとめて実行する.

2 目的の BASIC プログラムをロードする.

```
LOAD"PROGRAM"
```

3 ID を作成する.

```
AD=&H8000
FOR I=0 TO 31      ┐
  POKE AD+I, 0     │ IDエリアのクリアー
NEXT I             ┘
POKE &H8000,ASC("A")
POKE &H8001,ASC("B")
POKE &H8008,&H20
POKE &H8009,&H80
```

4 8000H～BFFFH をそのまま ROM に焼き込む.

7.3.2　カートリッジ用ソフトの作成に関する諸注意

●カートリッジで使用するワークエリアの確保

他のカートリッジに入っているプログラムといっしょに実行する必要がないプログラム(ゲームカートリッジのようなスタンドアロンのソフトウェア)では，BIOSの使用するワークエリア(F380H)よりアドレスの小さい部分は自由に使用することができます．

しかし，BASICインタープリタの機能を利用して実行されるプログラムでは，同じ領域をワークエリアとして使用するわけにはいきません．その対策としては以下の3つの方法があります．

(1)カートリッジ自体にRAMをのせてしまう(最も安全確実な方法)．

(2)必要なワークエリアが1バイトあるいは2バイトならばSLTWRK(FD09H～)の自分に対応する2バイトをワークとして使用する．

(3)必要なワークエリアが3バイト以上ならばBASICで使用するRAMから確保する．具体的には，まずBOTTOM(FC48H)の内容をSLTWRK(FD09H～)の対応するエリアに書き写し，BOTTOMの値を必要なワークエリア分増やし，そこをワークエリアとして確保する(図5.52)．

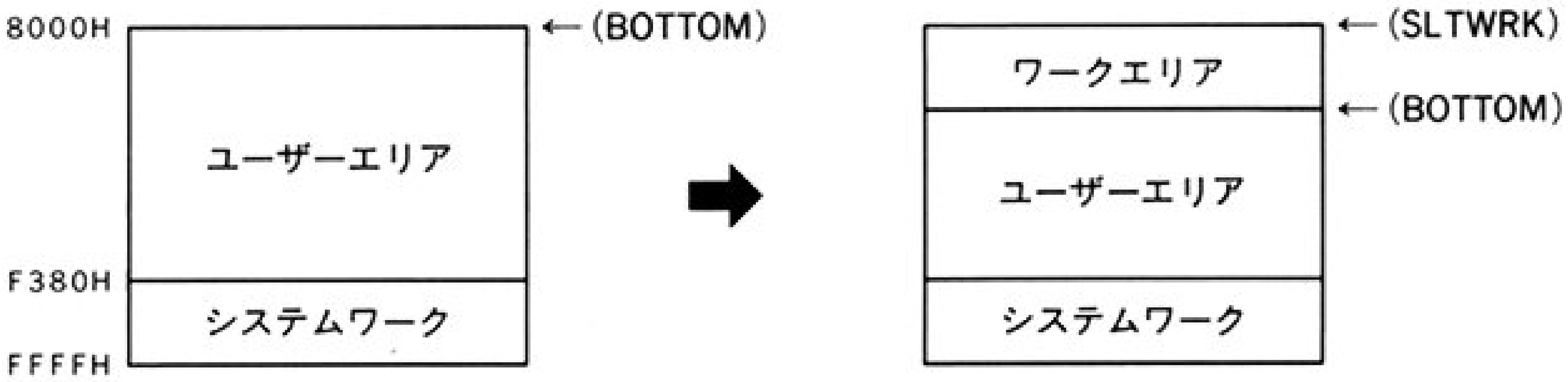


図5.52　ワークエリアの確保

(2)と(3)については以下のリストを参考にしてください．

リスト5.13　ワークエリア確保の実行例

```
;=======================================================
:  List 5.13   subroutines to support slot
;               for ROM in 1 page
;=======================================================

RSLREG   EQU     0138H
EXPTBL   EQU     0FCC1H
BOTTOM   EQU     0FC48H
HIMEM    EQU     0FC4AH
SLTWRK   EQU     0FD09H

;-----------------------------------------------------------
;
:        GTSL1   Get slot number of designated page
:        Entry   None
:        Return  A       Slot address as follows
:        Modify  Flags
```

```
;
;               FxxxSSPP
;               |   ||||
;               |   ||++-- primary slot # (0-3)
;               |   ++---- secondary slot # (0-3)
;               |          00 if not expanded
;               +--------- 1 if secondary slot # specified
;
;       This  value  can  later  be  used  as  an  input
;       parameter  for  the RDSLT, WRSLT, CALSLT, ENASLT
;       and 'RST 30H'
;
        PUBLIC  GTSL1@
GTSL1@:
        PUSH    HL              ;Save registers
        PUSH    DE
;
        CALL    RSLREG          ;read primary slot #
        RRCA
        RRCA
        AND     11B             ;[A]=000000PP
        LD      E,A
        LD      D,0             ;[DE]=000000PP
        LD      HL,EXPTBL
        ADD     HL,DE           ;[HL]=EXPTBL+000000PP
        LD      E,A             ;[E]=000000PP
        LD      A,(HL)          ;[A]=(EXPTBL+000000PP)
        AND     80H             ;Use only MSB
        JR      Z,GTSL1NOEXP
        OR      E               ;[A]=F00000PP
        LD      E,A             ;save primary slot number
        INC     HL              ;point to SLTTBL entry
        INC     HL
        INC     HL
        INC     HL
        LD      A,(HL)          ;get currently expansion slot register
        RRCA
        RRCA
        AND     11B             ;[A] = 000000SS
        RLCA
        RLCA                    ;[A] = 0000SS00
        OR      E               ;[A] = F000SSPP
;
GTSL1END:
        POP     DE
        POP     HL
        RET
GTSL1NOEXP:
        LD      A,E             ;[A] = 000000PP
        JR      GTSL1END

;-----------------------------------------------------------------
;
;       ASLW1   Get address of slot work
;       Entry   None
;       Return  HL      address of slot work
;       Modify  None
;
        PUBLIC  ASLW1@
ASLW1@:
        PUSH    DE
        PUSH    AF
        CALL    GTSL1@          ;[A] = F000SSPP, SS = 00 if not expanded
        AND     00001111B       ;[A] = 0000SSPP
        LD      L,A             ;[L] = 0000SSPP
        RLCA
        RLCA
```

```
        RLCA
        RLCA                            ;[A] = SSPP0000
        AND     00110000B               ;[A] = 00PP0000
        OR      L                       ;[A] = 00PPSSPP
        AND     00111100B               ;[A] = 00PPSS00
        OR      01B                     ;[A] = 00PPSSBB
;
;       Now, we have the sequence number for this
;       cartridge as follows.
;
;       00PPSSBB
;         ||||||
;         ||||++-- higher 2 bits of memory address (1)
;         ||++---- secondary slot # (0..3)
;         ++------ primary slot # (0..3)
;
        RLCA                            ;*= 2
        LD      E,A
        LD      D,0                     ;[DE] = 0PPSSBB0
        LD      HL,SLTWRK
        ADD     HL,DE
        POP     AF
        POP     DE
        RET

;-------------------------------------------------------
;
;       RSLW1   Read slot work
;       Entry   None
;       Return  HL      Content of slot work
;       Modify  None
;
        PUBLIC  RSLW1@
RSLW1@:
        PUSH    DE
        CALL    ASLW1@                  ;[HL] = address of slot work
        LD      E,(HL)
        INC     HL
        LD      D,(HL)                  ;[DE] = (slot work)
        EX      DE,HL                   ;[HL] = (slot work)
        POP     DE
        RET

;-------------------------------------------------------
;
;       WSLW1   Write slot work
;       Entry   HL      Data to write
;       Return  None
;       Modify  None
;
        PUBLIC  WSLW1@
WSLW1@:
        PUSH    DE
        EX      DE,HL                   ;[DE] = data to write
        CALL    ASLW1@                  ;[HL] = address of slot work
        LD      (HL),E
        INC     HL
        LD      (HL),D
        EX      DE,HL                   ;[HL] = data to write
        POP     DE
        RET

;-------------------------------------------------------
;
; How to allocate work area for cartridges
; If the work area is greater than 2 bytes, make the SLTWRK
```

```
; point to the system variable  BOTTOM  (0FC48H),  then  update
; it by  the  amount  of  memory required.  BOTTOM is set up by
; the initialization code to point to the  bottom  of  equipped
; RAM.
;           Ex. if the program is at 4000H..7FFFH.
;
;       WORKB   allocate work area from BOTTOM
;               (my slot work) <- (old BOTTOM)
;       Entry   HL      required memory size
;       Return  HL      start address of my work erea = old BOTTOM
;                       0 if can not allocate
;       Modify  None
;
        PUBLIC  WORKB@
WORKB@:
        PUSH    DE
        PUSH    BC
        PUSH    AF
;
        EX      DE,HL           ;[DE] = Size
        LD      HL,(BOTTOM)     ;Get current RAM bottom
        CALL    WSLW1@          ;Save BOTTOM to slot work
        PUSH    HL              ;Save old BOTTOM
        ADD     HL,DE           ;[HL] = (BOTTOM) + SIZE
        LD      A,H             ;Beyond 0DFFFH?
        CP      0E0H
        JR      NC,NOROOM       ;Yes, cannot allocate this much
        LD      (BOTTOM),HL     ;Update (BOTTOM)
        POP     HL              ;[HL] = old BOTTOM
WORKBEND:
        POP     AF
        POP     BC
        POP     DE
        RET
;
;       BOTTOM became  greater  than  0DFFFH,  there  is
;       no RAM left to be allocated.
;
NOROOM:
        LD      HL,0
        CALL    WSLW1@          ;Clear slot work
        JR      WORKBEND        ;Return 0 in [HL]

        END
```

●フック

MSX-BASICが使用するワークエリアのFD9AH～FFC9Hには“フック”というBASICの機能拡張のための領域があります．1つのフックは5バイトあり，そこには普通“RET”が書かれています．

MSX-BASICはある処理(ワークエリアのフックの説明で示すような処理)を行うと，そこから一度このフックをコールします．“RET”の場合はすぐ戻ってしまいますが，その5バイトをあらかじめイニシャライズ(INIT)ルーチンによりカートリッジ内のプログラムにインタースロットコールするよう書き換えておけば，BASICの機能を拡張することができるわけです(図5.53参照)．

この例として，割り込み処理用のH.KEYIというフックをカートリッジが利用する場合のプログラムを**リスト5.14**に示します

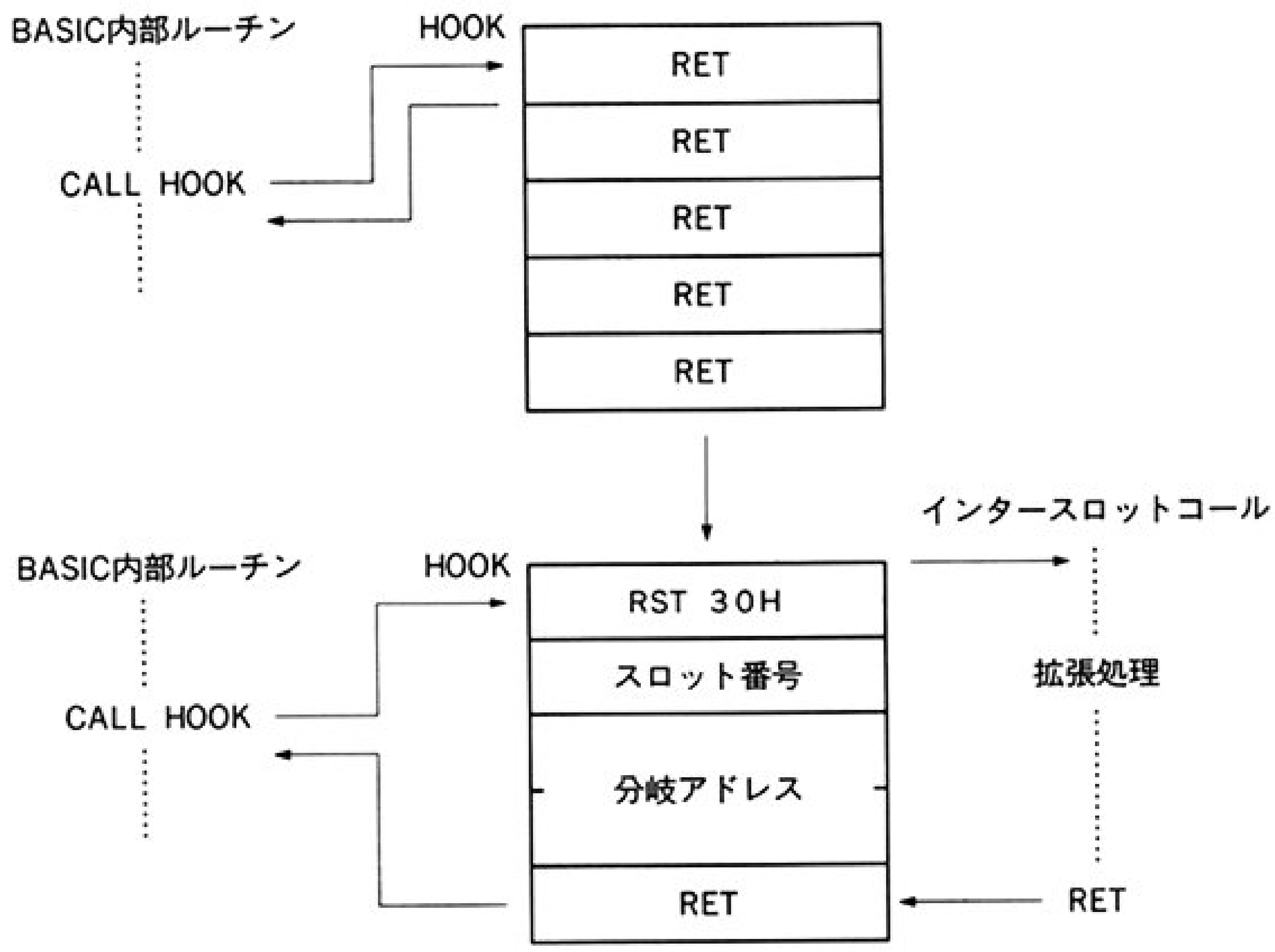


図5.53　フックの設定

リスト5.14　フックの使用

```
;============================================================
;  List 5.14  Sample program to use HOOK
;============================================================
;
; Start-up initialize entry
; This procedure will be called when system initializing.
;
H.KEYI   EQU     0FD9AH          ; interrupt hook
EXPTBL   EQU     0FCC1H          ; slots expanded or not
PSLTRG   EQU     0A8H            ; I/O port address of primary slot register
EXT      MYINT                   ; my interrrupt handler

INIT:
;        <<< Please insert other initialize routine here. if you need. >>>

; Set interrupt entry

         DI                      ; start of critical region

; Get old interrupt entry inter-slot call hook

         LD      DE,OLDINT       ; set address of old interrupt hook saved area
         LD      HL,H.KEYI       ; set address of interrupt entry hook
         LD      BC,5            ; length of hook is 5 bytes
         LDIR                    ; transfer
```

```
; What slot address is this cartridge placed

        CALL    GTMSLT          ; get my slot address

; Set new inter-slot call of interrupt entry

        LD      (H.KEYI+1),A    ; set slot address
        LD      A,0F7H          ; `RST 30H' inter-slot call operation code
        LD      (H.KEYI),A      ; set new hook op-code
        LD      HL,INTENT       ; get our interrupt entry point
        LD      (H.KEYI+2),HL   ; set new interrupt entry point
        LD      A,0C9H          ; `RET' operation code
        LD      (H.KEYI+4),A    ; set operation code of `RET'
        EI                      ; end of critical region
        RET

;----------------------------------------------------------
; What slot address is this cartridge placed
;  Entry: No
;  Action: Compute my slot address
;  Return: A = slot address
;  Modify: Flag

GTMSLT:
        PUSH    BC              ; save environment
        PUSH    HL
        IN      A,(PSLTRG)      ; read primary slot register
        RRCA                    ; move it to bit 0,1 of A
        RRCA
        AND     00000011B       ; get bit 1,0
        LD      C,A             ; set primary slot No.
        LD      B,0
        LD      HL,EXPTBL       ; see is this slot is expanded or not
        ADD     HL,BC
        OR      (HL)            ; set MSB if so
        LD      C,A
        INC     HL              ; point to SLTTBL entry
        INC     HL
        INC     HL
        INC     HL
        LD      A,(HL)          ; get  what  is  currently  output to expansion
                                ; slot register

        AND     00001100B       ; get bit 3,2
        OR      C               ; finely form slot address

        POP     HL              ; restore environment
        POP     BC
        RET                     ; return to main

;----- Interrupt entry -----

INTENT:
        CALL    MYINT           ; call interrupt handler
        JP      OLDINT          ; go old interrupt handler

;----- HOOK save area -----

OLDINT: DS      5

        END
```

●スタックポインタの初期化

ディスク内蔵のMSXの場合，スロットの位置によってはカートリッジよりもディスク・インターフェイスROMの方が先に初期化動作を行い，ワークエリア確保のためスタックポインタを下位アドレス方向に下げることがあります．この時，ディスクを使わないソフトウェアはカートリッジに制御が渡ってきてからもう一度スタックポインタを設定し直さないと，スタック領域がなくなり暴走などのドラブルを生じる可能性があります．プログラムの初めにはスタックポインタのイニシャライズを忘れずに行ってください．

●拡張スロットでの動作チェック

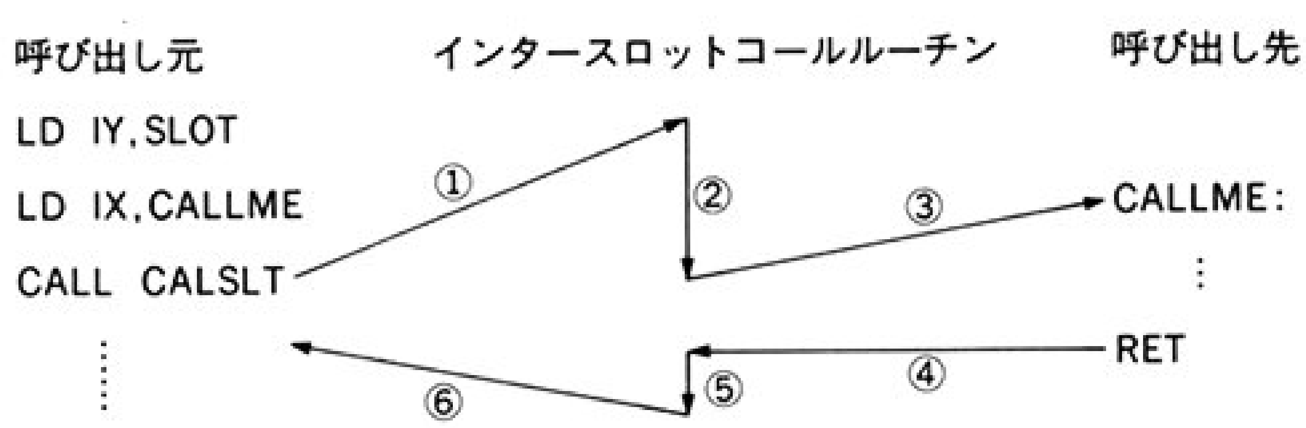
一般の市販ソフトウェアが拡張スロットに差し込まれた場合やRAMが拡張スロットにある場合に，アプリケーションプログラムが動作しないという問題があります．MSX 2の多くの機種は本体内部で拡張スロットを使用しているので，拡張スロットでの動作不良は致命的です．市販されるソフトウェアは，それが拡張スロットに入れられている場合とRAMが拡張スロットにある場合について，かならず動作チェックを行ってください．

特にFFFFH番地には拡張スロットレジスタが置かれますので，そこをRAMのつもりで参照してはいけません．たとえば，プログラム中で“LD SP, 0”を行ってFFFFH番地にスタックを設定すると，そのプログラムは拡張スロットを用いた機種では暴走します．

● CALSLT使用時の注意

CALSLTおよびCALLFでインタースロットコールを行うと，IX，IY，および裏レジスタの内容は破壊されます．また，このルーチンから帰ってくる時，MSX 1では割り込みが禁止されていますが，MSX 2では呼び出す前の状態に戻ります．

CALSLTまたはCALLFを使ってインタースロットコールを行うと，目的のプログラムが呼ばれる時(図の③)と呼び出し元に戻る時(図の⑥)に必ず割り込みが禁止されています．



MSX2では，インタースロットコールルーチンの前後で割り込みの状態が保存されます．つまり，図の③の時に①と同じ状態になり，⑥の時に④と同じ状態になります．呼び出されたプログラムが“EI”や“DI”を行う場合は，注意して下さい．

APPENDIX

A.1 BIOS一覧

ここではユーザーが利用可能な126のBIOSエントリを紹介します.

BIOSにはMAIN-ROMとSUB-ROM内のルーチンの2種類あり,コーリング・シーケンスはそれぞれ異なります(これについては後述). ここでエントリは, 次のように表記します.

ラベル名(アドレス)　*n

機能	機能解説および注意
入力	呼び出し時に必要なパラメータ
出力	リターンされるパラメータ
レジスタ	内容が破壊されるレジスタ

nの値は次のような意味を持ちます.

*1………MSX1と同じもの.

*2………スクリーン・モード5～8では内部でSUB-ROMを呼び出す.

*3………つねにSUB-ROMを呼び出す.

*4………スクリーン・モード4～8の変更はあるが, SUB-ROMは呼び出さない.

また, 何もついていないものは, MSX2用に新たに追加されたルーチンです.

MAIN-ROM

MAIN-ROM内のルーチンを呼び出す場合は, 通常のサブルーチンコールとしてCALL命令またはRST命令で行います.

● RST関係

以下のRSTのうち, RST 00H～RST 28HはBASICインタープリタが使います. RST 30Hはスロット間コール(インタースロット・コール)に, RST 38Hはハードウェア割り込みに使われます.

CHKRAM (0000H)　*1

機能	RAMをチェックし,システム用のRAMスロットをセット
入力	なし
出力	なし
レジスタ	すべて

SYNCHR (0008H)　*1

機能	〔HL〕の1文字が指定した文字かどうかチェック. 違っていたらSYNTAX ERRORを発生し,同じならCHRGTR(0010H)へ行く
入力	〔HL〕にチェックする文字, 比較する文字をこのルーチンを呼び出すRST命令の次にセット(インライン・パラメータ)

例

```
        LD      HL, MOJI
        RST     08H
        DB      'A'
                 :
                 :
                 :
MOJI:   DB      'B'
```

出力　HLが1進み，Aには〔HL〕が入る．チェックした文字が数字であればCYフラグをセット，ステートメントの終わり(00Hまたは3AH)ならばZフラグをセット
レジスタ　AF，HL

RDSLT (000CH)　*1

機能　Aの値に対応するスロットを選択し，そのスロットのメモリを1バイト読む．このルーチンを呼ぶと割り込みを禁止し，実行を終えても解除しない
入力　Aにスロット番号

```
〔F000EEPP〕
 |   |  |
 |   |  └──── 基本スロット番号
 |   |        (0～3)
 |   └─────── 拡張スロット番号
 |            (0～3)
 └─────────── 拡張スロットを指
              定するとき“1”
```

HLに読み込むメモリの番地
出力　Aに読み込んだメモリの値
レジスタ　AF，BC，DE

CHRGTR (0010H)　*1

機能　BASICテキストから文字（またはトークン）を取り出す
入力　〔HL〕に読み込む文字
出力　HLが1進み，Aには〔HL〕が入る．文字が数字であればCYフラグをセット，ステートメントの終わりならばZフラグをセット
レジスタ　AF，HL

WRSLT (0014H)　*1

機能　Aの値に対応するスロットを選択し，そのスロットのメモリに値を1バイト書き込む．このルーチンを呼ぶと割り込みを禁止し，実行を終えても解除しない
入力　Aでスロットを指定（RDSLTと同じ）．HLに書き込む番地，Eにその値
出力　なし
レジスタ　AF，BC，D

OUTDO (0018H)　*2

機能　現在使っているデバイスに値を出力
入力　Aに出力する値
PRTFLG (F416H) が0以外ならプリンタ出力
PTRFIL(F864H)が0以外ならPTRFILで示されるファイルへ出力
出力　なし
レジスタ　なし

CALSLT (001CH)　*1

機能　他のスロットのルーチンを呼び出す（インタースロット・コール）．
入力　IYレジスタの上位8ビットでスロットを指定(RDSLTと同じ)．IXにコールする番地
出力　呼び出すルーチンによる
レジスタ　呼び出すルーチンによる

DCOMPR (0020H)　*1

機能　HLとDEの内容を比較
入力　HL，DE
出力　HL＝DEならばZフラグを，HL<DEならCYフラグをセット
レジスタ　AF

ENASLT (0024H)　*1

機能　Aの値に対応するスロットを選択し，以降そのスロットを使用可能にする．このルーチンを呼ぶと割り込みを禁止し，実行を終えても解除しない
入力　Aでスロットを指定（RDSLTと同じ）
出力　なし
レジスタ　すべて

GETYPR (0028H)　*1

機能　DAC（デシマル・アキュムレータ）の型を返す
入力　VALTYP (F663H) にはDACの型が入っている
出力　DACの型によってS，Z，P/V，CYフラグが以下のように変化する

・整数型	・単精度実数型
C = 1	C = 1
S = 1　*	S = 0
Z = 0	Z = 0
P/V = 1	P/V = 0　*

・文字列型　　・倍精度実数型
C＝1　　C＝0 ＊
S＝0　　S＝0
Z＝1 ＊　　Z＝0
P/V＝1　　P/V＝1
各型は，＊マークの付いたフラグを調べればチェックできる
レジスタ AF

CALLF (0030H)　＊1
機能 他のスロットのルーチンを呼び出す．呼び出し方は次のとおり

```
RST  30H
DB   n    ; nはスロット番号
          (RDSLTと同じ)
DW   nn   ; nnは呼び出すアドレス
```

入力 上記の方法による
出力 呼び出すルーチンによる
レジスタ AF，その他は呼び出すルーチンによる

KEYINT (0038H)　＊1
機能 タイマ割り込みの処理ルーチンを実行
入力 なし
出力 なし
レジスタ なし

● I/O 初期化

INITIO (003BH)　＊1
機能 デバイスを初期化
入力 なし
出力 なし
レジスタ すべて

INIFNK (003EH)　＊1
機能 ファンクションキーの内容を初期化
入力 なし
出力 なし
レジスタ すべて

● VDP アクセス用

DISSCR (0041H)　＊1
機能 画面表示の禁止
入力 なし
出力 なし
レジスタ AF，BC

ENASCR (0044H)　＊1
機能 画面の表示
入力 なし
出力 なし
レジスタ AF，BC

WRTVDP (0047H)　＊2
機能 VDPのレジスタにデータを書き込む
入力 Cにレジスタ番号，Bにデータ．レジスタ番号は0～23および32～46
出力 なし
レジスタ AF，BC

RDVRM (004AH)　＊1
機能 VRAMの内容を読む．ただし，これはTMS9918に対するものでVRAMのアドレスは下位14ビットのみ有効．全ビットを使う場合はNRDVRMをコール
入力 HLに読み出すVRAMのアドレス
出力 Aに読み出した値
レジスタ AF

WRTVRM (004DH)　＊1
機能 VRAMにデータを書き込む．ただし，これはTMS9918に対するものでVRAMのアドレスは14ビットのみ有効．全ビットを使う場合はNWRVRMをコール
入力 HLにVRAMアドレス，Aにデータ
出力 なし
レジスタ AF

SETRD (0050H)　＊1
機能 VDPにVRAMアドレスをセットして，読み出せる状態にする．これは，VDPのアドレス・オート・インクリメントの機能を使って，連続したVRAM領域からデータを読み出す時に使用する．これによりRDVRMをループ中で使うより高速に読み出しが可能になる．ただし，これはTMS9918に対するものでVRAMのアドレスは14ビットのみ有効．全ビットを使う場合はNSETRDをコール
入力 HLにVRAMアドレス
出力 なし
レジスタ AF

SETWRT (0053H)　*1

機能　VDPにVRAMアドレスをセットして，書き込める状態にする．使用目的はSETRDと同様．ただし，これはTMS9918に対するものでVRAMのアドレスは14ビットのみ有効．全ビットを使う場合はNSTWRTをコール

入力　HLにVRAMアドレス

出力　なし

レジスタ　AF

FILVRM (0056H)　*4

機能　VRAMの指定領域を同一のデータでうめる．ただし，これはTMS9918に対するものでVRAMのアドレスは14ビットのみ有効．全ビットを使う場合はBIGFILを参照

入力　HLに書き込みを開始するVRAMアドレス，BCに書き込む領域の長さ，Aにデータ

出力　なし

レジスタ　AF，BC

LDIRMV (0059H)　*4

機能　VRAMからメモリへブロック転送

入力　HLにソース・アドレス(VRAM)，DEにデスティネーション・アドレス(メモリ)，BCに長さ．指定するVRAMのアドレスは全ビット有効

出力　なし

レジスタ　すべて

LDIRVM (005CH)　*4

機能　メモリからVRAMへブロック転送

入力　HLにソース・アドレス(メモリ)，DEにデスティネーション・アドレス(VRAM)，BCに長さ．指定するVRAMのアドレスは全ビット有効

出力　なし

レジスタ　すべて

CHGMOD (005FH)　*3

機能　スクリーン・モードを変える．パレットは初期化しない．パレットの初期化が必要ならSUB-ROMのCHGMDPを参照

入力　Aにスクリーン・モード（0～8）

出力　なし

レジスタ　すべて

CHGCLR (0062H)　*1

機能　画面の色を変える

入力　Aにモード
FORCLR（F3E9H）に前景色
BAKCLR（F3EAH）に背景色
BDRCLR（F3EBH）に周辺色

出力　なし

レジスタ　すべて

NMI (0066H)　*1

機能　NMI（Non Maskable Interrupt）処理ルーチンを実行

入力　なし

出力　なし

レジスタ　なし

CLRSPR (0069H)　*3

機能　すべてのスプライトを初期化．スプライト・パターンをヌルに，スプライト番号をスプライト面番号に，スプライト・カラーを前景色にする．スプライトの垂直位置は209（モード0～3）または217（モード4～8）にセット

入力　SCRMOD（FCAFH）にスクリーン・モード

出力　なし

レジスタ　すべて

INITXT (006CH)　*3

機能　画面をTEXT1モード（40＊24）に初期化．このルーチンではパレットは初期化しない．パレットの初期化が必要なら，このルーチンをコールした後，SUB-ROMのINIPLTをコールする

入力　LINL40(F3AEH)に1行の幅

TXTNAM (F3B3H) にパターンネーム・テーブル
TXTCGP (F3B7H) にパターンジェネレータ・テーブル

出力 なし
レジスタ すべて

INIT32 (006FH) *3

機能 画面をGRAPHIC1モード(32*24)に初期化. このルーチンではパレットは初期化しない
入力 T32NAM (F3BDH) にパターンネーム・テーブル
T32COL (F3BFH) にカラーテーブル
T32CGP (F3C1H) にパターンジェネレータ・テーブル
T32ATR (F3C3H) にスプライトアトリビュート・テーブル
T32PAT (F3C5H) にスプライトジェネレータ・テーブル
出力 なし
レジスタ すべて

INIGRP (0072H) *3

機能 画面を高解像グラフィックモードに初期化. このルーチンではパレットは初期化しない
入力 GRPNAM (F3C7H) にパターンネーム・テーブル
GRPCOL (F3C9H) にカラーテーブル
GRPCGP (F3CBH) にパターンジェネレータ・テーブル
GRPATR (F3CDH) にスプライトアトリビュート・テーブル
GRPPAT (F3CFH) にスプライトジェネレータ・テーブル
出力 なし
レジスタ すべて

INIMLT (0075H) *3

機能 画面をMULTI COLORモードに初期化. このルーチンではパレットは初期化しない
入力 MLTNAM (F3D1H) にパターンネーム・テーブル
MLTCOL (F3D3H) にカラーテーブル
MLTCGP (F3D5H) にパターンジェネレータ・テーブル
MLTATR (F3D7H) にスプライトアトリビュート・テーブル
MLTPAT (F3D9H) にスプライトジェネレータ・テーブル
出力 なし
レジスタ すべて

SETTXT (0078H) *3

機能 VDPのみをTEXT1モード(40*24)にする
入力 INITXTと同じ
出力 なし
レジスタ すべて

SETT32 (007BH) *3

機能 VDPのみをGRAPHIC1モード (32*24) にする
入力 INIT32と同じ
出力 なし
レジスタ すべて

SETGRP (007EH) *3

機能 VDPのみをGRAPHIC2モードにする
入力 INIGRPと同じ
出力 なし
レジスタ すべて

SETMLT (0081H) *3

機能 VDPのみをMULTI COLORモードにする
入力 INIMLTと同じ
出力 なし
レジスタ すべて

CALPAT (0084H) *1

機能 スプライト・ジェネレータ・テーブルのアドレスを返す
入力 Aにスプライト番号
出力 HLにアドレス
レジスタ AF, DE, HL

CALATR (0087H) *1

機能 スプライトアトリビュート・テーブルのアドレスを返す
入力 Aにスプライト番号
出力 HLにアドレス
レジスタ AF, DE, HL

GSPSIZ (008AH)　*1

機能	現在のスプライト・サイズを返す
入力	なし
出力	Aにスプライト・サイズ（バイト数）．16＊16のサイズの場合のみCYフラグをセット．それ以外の場合はCYフラグをリセット
レジスタ	AF

GRPPRT (008DH)　*2

機能	グラフィック画面に文字を表示
入力	Aにキャラクタコード．スクリーン・モードが5～8ならLOGOPR(FB02H)にロジカル・オペレーションコード
出力	なし
レジスタ	なし

● PSG関連

GICINI (0090H)　*1

機能	PSGを初期化し，PLAY文のための初期値をセット
入力	なし
出力	なし
レジスタ	すべて

WRTPSG (0093H)　*1

機能	PSGのレジスタにデータを書き込む
入力	AにPSGのレジスタ番号，Eにデータ
出力	なし
レジスタ	なし

RDPSG (0096H)　*1

機能	PSGのレジスタの値を読む
入力	AにPSGのレジスタ番号
出力	Aに読み込んだ値
レジスタ	なし

STRTMS (0099H)　*1

機能	バックグラウンド・タスクとしてPLAY文が実行中であるかどうかチェックし，実行中でなければPLAY文の実行を開始
入力	なし
出力	なし
レジスタ	すべて

●キーボード，CRT，プリンタへの入出力

CHSNS (009CH)　*1

機能	キーボード・バッファの状態をチェック
入力	なし
出力	バッファが空であればZフラグをセット．そうでなければZフラグをリセット
レジスタ	AF

CHGET (009FH)　*1

機能	1文字入力（入力待ちあり）
入力	なし
出力	Aに入力された文字コード
レジスタ	AF

CHPUT (00A2H)　*1

機能	1文字表示
入力	Aに出力する文字コード
出力	なし
レジスタ	なし

LPTOUT (00A5H)　*1

機能	1文字プリンタ出力
入力	Aに出力する文字コード
出力	失敗した場合はCYフラグをセット
レジスタ	F

LPTSTT (00A8H)　*1

機能	プリンタの状態をチェック
入力	なし
出力	Aが255でZフラグがリセットされていればプリンタはREADY．Aが0でZフラグがセットされていればプリンタはNOT READY
レジスタ	AF

CNVCHR (00ABH)　*1

機能	グラフィック・ヘッダかどうかをチェックし，コードを変換
入力	Aに文字コード
出力	CYフラグがリセット→グラフィック・ヘッダではない CYフラグとZフラグがセット→Aに変換後のコード CYフラグがセット，Zフラグがリセット→Aに変換されていないコード
レジスタ	AF

PINLIN (00AEH)　*1

機能　リターンキーやSTOPキーがタイプされるまでに入力された文字コードを指定されたバッファに格納する
入力　なし
出力　HLにバッファの先頭アドレス−1，STOPキーで終了したときのみCYフラグをセット
レジスタ　すべて

INLIN (00B1H)　*1

機能　AUTFLG (F6AAH) がセットされる以外はPINLINと同じ
入力　なし
出力　HLにバッファの先頭アドレス−1，STOPキーで終了したときのみCYフラグをセット
レジスタ　すべて

QINLIN (00B4H)　*1

機能　“？”とスペース1個を表示してINLINを実行
入力　なし
出力　HLにバッファの先頭アドレス−1，STOPキーで終了したときのみCYフラグをセット
レジスタ　すべて

BREAKX (00B7H)　*1

機能　Ctrl-STOPキーを押しているかどうかチェック．このルーチンでは割り込みが禁止される
入力　なし
出力　押されていればCYフラグをセット
レジスタ　AF

BEEP (00C0H)　*3

機能　ブザーを鳴らす
入力　なし
出力　なし
レジスタ　すべて

CLS (00C3H)　*3

機能　画面クリア
入力　ゼロフラグをセット
出力　なし
レジスタ　AF，BC，DE

POSIT (00C6H)　*1

機能　カーソルの移動
入力　HにカーソルのX座標，LにカーソルのY座標
出力　なし
レジスタ　AF

FNKSB (00C9H)　*1

機能　ファンクション・キーの表示がアクティブかどうかチェックし (FNKFLG)，アクティブなら表示，でなければ消す
入力　FNKFLG (FBCEH)
出力　なし
レジスタ　すべて

ERAFNK (00CCH)　*1

機能　ファンクション・キーの表示を消す
入力　なし
出力　なし
レジスタ　すべて

DSPFNK (00CFH)　*2

機能　ファンクション・キーを表示
入力　なし
出力　なし
レジスタ　すべて

TOTEXT (00D2H)　*1

機能　画面を強制的にテキストモードにする
入力　なし
出力　なし
レジスタ　すべて

●ゲームI/Oアクセス

GTSTCK (00D5H)　*1

機能　ジョイスティックの状態を返す
入力　Aに調べるジョイスティックの番号
出力　Aにジョイスティックの方向
レジスタ　すべて

GTTRIG (00D8H)　*1

機能　トリガボタンの状態を返す
入力　Aに調べるトリガボタンの番号
出力　Aが0ならトリガボタンは押されていない
Aが0FFHならトリガボタンは押されている

レジスタ　AF

GTPAD (00DBH)　*1
機能　タッチパッドの状態を返す
入力　Aに調べるタッチパッドの番号
出力　Aに値
レジスタ　すべて

GTPDL (00DEH)　*2
機能　パドルの値を返す
入力　Aにパドルの番号
出力　Aに値
レジスタ　すべて

●カセット入出力ルーチン

TAPION (00E1H) *1
機能　カセットのモーターONの後，ヘッダ・ブロックを読む
入力　なし
出力　失敗した場合はCYフラグをセット
レジスタ　すべて

TAPIN (00E4H)　*1
機能　テープからデータを読む
入力　なし
出力　Aにデータ．失敗した場合はCYフラグをセット
レジスタ　すべて

TAPIOF (00E7H)　*1
機能　テープからの読み込みをストップ
入力　なし
出力　なし
レジスタ　なし

TAPOON (00EAH)　*1
機能　カセットのモーターONの後，ヘッダ・ブロックを書き込む
入力　A＝0ならショート・ヘッダ
A≠0ならロング・ヘッダ
出力　失敗した場合はCYフラグをセット
レジスタ　すべて

TAPOUT (00EDH)　*1
機能　テープにデータを書き込む
入力　Aにデータ
出力　失敗した場合はCYフラグをセット
レジスタ　すべて

TAPOOF (00F0H)　*1
機能　テープへの書き込みをストップ
入力　なし
出力　失敗した場合はCYフラグをセット
レジスタ　なし

STMOTR (00F3H)　*1
機能　カセットのモーターの動作設定
入力　A＝0　→　ストップ
A＝1　→　スタート
A＝0FFH　→　現在と逆の動作
出力　なし
レジスタ　AF

●その他

CHGCAP (0132H)　*1
機能　CAPランプの状態を変える
入力　A＝0　→　ランプを消す
A≠0　→　ランプをつける
出力　なし
レジスタ　AF

CHGSND (0135H)　*1
機能　1ビット・サウンドポートの状態を変える
入力　A＝0　→　OFF
A≠0　→　ON
出力　なし
レジスタ　AF

RSLREG (0138H)　*1
機能　基本スロット・レジスタに現在出力している内容を読む
入力　なし
出力　Aに読み込んだ値
レジスタ　A

WSLREG (013BH)　*1
機能　基本スロット・レジスタにデータを書き込む
入力　Aに書き込む値
出力　なし
レジスタ　なし

RDVDP (013EH) *1

機能 VDPのステータス・レジスタを読む
入力 なし
出力 Aに読み込んだ値
レジスタ A

SNSMAT (0141H) *1

機能 キーボード・マトリクスから指定した行の値を読む
入力 Aに指定する行
出力 Aにデータ（押されているキーに対応するビットが0になる）
レジスタ AF, C

ISFLIO (014AH) *1

機能 デバイスが動作中かどうかチェック
入力 なし
出力 A＝0 → 動作中
A≠0 → 動作中ではない
レジスタ AF

OUTDLP (014DH) *1

機能 プリンタ出力．LPTOUTとは次の点で異なる
1. TABはスペースに展開される
2. MSX仕様でないプリンタの場合，ひらがなをカタカナに，グラフィック文字を1バイト文字に変換する
3. 失敗した場合，device I/O errorとなる

入力 Aにデータ
出力 なし
レジスタ F

KILBUF (0156H) *1

機能 キーボード・バッファをクリア
入力 なし
出力 なし
レジスタ HL

CALBAS (0159H) *1

機能 BASICインタープリタ内のルーチンをインタースロット・コール
入力 IXに呼び出すアドレス
出力 呼び出すルーチンによる
レジスタ 呼び出すルーチンによる

● MSX2用に追加されたエントリ

SUBROM (015CH)

機能 SUB-ROMをインタースロット・コール
入力 IXに呼び出すアドレス，同時にIXをスタックに積む
出力 呼び出すルーチンによる
レジスタ 裏レジスタとIYはリザーブされる

EXTROM (015FH)

機能 SUB-ROMをインタースロット・コール
入力 IXに呼び出すアドレス
出力 呼び出すルーチンによる
レジスタ 裏レジスタとIYはリザーブされる

EOL (0168H)

機能 行の終わりまでデリート
入力 HにカーソルのX座標，LにY座標
出力 なし
レジスタ すべて

BIGFIL (016BH)

機能 機能的にはFILVRMと同じ．ただし，以下の点で異なる．
FILVRMでは，スクリーン・モードが0～3であるかをチェックし，もしそうならVDPは16KバイトのVRAMしか持っていないものとして扱う（MSX1とのコンパチビリティのため）．しかし，BIGFILはモードのチェックは行わず，与えられたパラメータどおりに動作する
入力 FILVRMと同じ
出力 FILVRMと同じ
レジスタ FILVRMと同じ

NSETRD (016EH)

機能 VDPにアドレスをセットして，読み込める状態にする
入力 HLにVRAMアドレス
出力 なし
レジスタ AF

NSTWRT (0171H)

機能 VDPにアドレスをセットして，書き込める状態にする
入力 HLにVRAMアドレス
出力 なし
レジスタ AF

NRDVRM (0174H)

機能　VRAMの内容を読む
入力　HLに読み出すVRAMのアドレス
出力　Aに読み出した値
レジスタ　F

NWRVRM (0177H)

機能　VRAMにデータを書き込む
入力　HLにVRAMアドレス，Aにデータ
出力　なし
レジスタ　AF

SUB-ROM

SUB-ROMのコーリング・シーケンスは以下のとおりです．

```
                 ⋮
        LD       IX, INIPLT
                         ; Set BIOS entry address
        CALL     EXTROM
                         ; Returns here
                 ⋮
```

また，IXを壊したくない場合には次のようにコールします．

```
                 ⋮
INIPAL: PUSH     IX
                         ; Save IX
        LD       IX, INIPLT
                         ; Set BIOS entry address
        JP       SUBROM
                         ; Returns caller of INIPAL
                 ⋮
```

GRPPRT (0089H)

機能　グラフィック画面に1文字出力（スクリーン5~8のみで動作）
入力　Aに文字コード
出力　なし
レジスタ　なし

NVBXLN (00C9H)

機能　ボックスを描く
入力　始点：BCにX座標，DEにY座標
終点：GXPOS (FCB3H) にX座標，
GYPOS (FCB5H) にY座標
色：ATRBYT(F3F2H)にアトリビュート
ロジカルオペレーション・コード
：LOGOPR (FB02H)
出力　なし
レジスタ　すべて

NVBXFL (00CDH)

機能　塗りつぶされたボックスを描く
入力　始点：BCにX座標，DEにY座標
終点：GXPOS (FCB3H) にX座標，
GYPOS (FCB5H) にY座標
色：ATRBYT (F3F2H) にアトリビュート
ロジカルオペレーション・コード
：LOGOPR (FB02H)
出力　なし
レジスタ　すべて

CHGMOD (00D1H)

機能　スクリーン・モードを変える
入力　Aにスクリーン・モード（0～8）
出力　なし
レジスタ　すべて

INITXT (00D5H)

機能　画面をテキスト・モード(40*24)にして初期化する
入力　TXTNAM (F3B3H) にパターンネーム・テーブル
TXTCGP(F3B7H)にパターンジェネレータ・テーブル
出力　なし
レジスタ　すべて

INIT32 (00D9H)

機能　画面をテキスト・モード（32*24）にして初期化する
入力　T32NAM (F3BDH) にパターンネーム・テーブル
T32COL (F3BFH) にカラーテーブル

T32CGP (F3C1H) にパターンジェネレータ・テーブル
T32ATR (F3C3H) にスプライトアトリビュート・テーブル
T32PAT (F3C5H) にスプライトジェネレータ・テーブル

出力 なし
レジスタ すべて

INIGRP (00DDH)
機能 画面を高解像グラフィック・モードにして初期化する
入力 GRPNAM (F3C7H) にパターンネーム・テーブル
GRPCOL (F3C9H) にカラーテーブル
GRPCGP (F3CBH) にパターンジェネレータ・テーブル
GRPATR (F3CDH) にスプライトアトリビュート・テーブル
GRPPAT (F3CFH) にスプライトジェネレータ・テーブル
出力 なし
レジスタ すべて

INIMLT (00E1H)
機能 画面をMULTI COLORモードにして初期化する
入力 MLTNAM (F3D1H) にパターンネーム・テーブル
MLTCOL (F3D3H) にカラーテーブル
MLTCGP (F3D5H) にパターンジェネレータ・テーブル
MLTATR (F3D7H) にスプライトアトリビュート・テーブル
MLTPAT (F3D9H) にスプライトジェネレータ・テーブル
出力 なし
レジスタ すべて

SETTXT (00E5H)
機能 VDPをテキスト・モード (40*24) にする
入力 INITXTと同じ
出力 なし
レジスタ すべて

SETT32 (00E9H)
機能 VDPをテキスト・モード (32*24) にする
入力 INIT32と同じ
出力 なし
レジスタ すべて

SETGRP (00EDH)
機能 VDPを高解像度モードにする
入力 INIGRPと同じ
出力 なし
レジスタ すべて

SETMLT (00F1H)
機能 VDPをMULTI COLORモードにする
入力 INIMLTと同じ
出力 なし
レジスタ すべて

CLRSPR (00F5H)
機能 すべてのスプライトを初期化．スプライト・パターンをヌルに，スプライト番号をスプライト面番号に，スプライトの色を前景色にする．スプライトの垂直位置は217にセット
入力 SCRMOD (FCAFH) にスクリーン・モード
出力 なし
レジスタ すべて

CALPAT (00F9H)
機能 スプライトジェネレータ・テーブルのアドレスを返す (このルーチンはMAIN-ROMの同名のBIOSと同じ)
入力 Aにスプライト番号
出力 HLにアドレス
レジスタ AF, DE, HL

CALATR (00FDH)
機能 スプライト属性テーブルのアドレスを返す (このルーチンはMAIN-ROMの同名のBIOSと同じ)
入力 Aにスプライト番号
出力 HLにアドレス
レジスタ AF, DE, HL

GSPSIZ (0101H)

機能 現在のスプライトサイズを返す（このルーチンはMAIN-ROMの同名のBIOSと同じ）
入力 なし
出力 Aにスプライトサイズ．16＊16のサイズの場合のみCYフラグがセット
レジスタ AF

GETPAT (0105H)

機能 キャラクタ・パターンを返す
入力 Aに文字コード
出力 PATWRK（FC40H）にキャラクタ・パターン
レジスタ すべて

WRTVRM (0109H)

機能 VRAMにデータを書き込む
入力 HLにVRAMアドレス（0～FFFFH），Aにデータ
出力 なし
レジスタ AF

RDVRM (010DH)

機能 VRAMの内容を読む
入力 HLに読み出すVRAMのアドレス（0～FFFFH）
出力 Aに読み出した値
レジスタ AF

CHGCLR (0111H)

機能 画面の色を変える
入力 Aにスクリーン・モード
FORCLR（F3E9H）に前景色
BAKCLR（F3EAH）に背景色
BDRCLR（F3EBH）に周辺色
出力 なし
レジスタ すべて

CLSSUB (0115H)

機能 画面クリア
入力 なし
出力 なし
レジスタ すべて

DSPFNK (011DH)

機能 ファンクション・キーの表示
入力 なし
出力 なし
レジスタ すべて

WRTVDP (012DH)

機能 VDPのレジスタにデータを書き込む
入力 Cにレジスタ番号，Bにデータ
出力 なし
レジスタ AF，BC

VDPSTA (0131H)

機能 VDPのレジスタを読む
入力 Aにレジスタ番号（0～9）
出力 Aにデータ
レジスタ F

SETPAG (013DH)

機能 ページの切り換え
入力 DPPAGE（FAF5H）にディスプレイ・ページ番号
ACPAGE（FAF6H）にアクティブ・ページ番号
出力 なし
レジスタ AF

INIPLT (0141H)

機能 パレットの初期化（現在のパレットはVRAMにセーブされている）
入力 なし
出力 なし
レジスタ AF，BC，DE

RSTPLT (0145H)

機能 パレットをVRAMからリストアする
入力 なし
出力 なし
レジスタ AF，BC，DE

GETPLT (0149H)

機能 パレットからカラーコードを得る
入力 Aにパレット番号(0～15)
出力 Bの上位4ビットに赤のコード
Bの下位4ビットに青のコード
Cの下位4ビットに緑のコード
レジスタ AF，DE

SETPLT (014DH)

機能　カラーコードをパレットにセット
入力　Dにパレット番号（0～15）
　Aの上位4ビットに赤のコード
　Aの下位4ビットに青のコード
　Eの下位4ビットに緑のコード
出力　なし
レジスタ　AF

BEEP (017DH)

機能　ブザーを鳴らす
入力　なし
出力　なし
レジスタ　すべて

PROMPT (0181H)

機能　プロンプトを表示
入力　なし
出力　なし
レジスタ　すべて

NEWPAD (01ADH)

機能　マウス，ライトペンの状態を読む
入力　Aに以下のデータを入れてコール．カッコ内は戻り値
　8 … ライトペン・チェック（0FFHで有効）
　9 … X座標を返す
　10 … Y座標を返す
　11 … ライトペン・スイッチの状態を返す（押されたとき0FFH）
　12 … マウスがポート1に接続されているか（0FFHで有効）
　13 … X方向のオフセットを返す
　14 … Y方向のオフセットを返す
　15 …（つねに0）
　16 … マウスがポート2に接続されているか（0FFHで有効）
　17 … X方向のオフセットを返す
　18 … Y方向のオフセットを返す
　19 …（つねに0）
出力　A
レジスタ　すべて

CHGMDP (01B5H)

機能　VDPのモードを変える．パレットは初期化される
入力　Aにスクリーン・モード(0～8)
出力　なし
レジスタ　すべて

KNJPRT (01BDH)

機能　グラフィック画面（モード5～8）に漢字出力
入力　BCにJIS漢字コード，Aに表示モード．表示モードはBASICのPUT KANJI命令と同様に以下の意味を持つ
　0 … 16×16ドットで表示
　1 … 偶数番目のドットを表示
　2 … 奇数番目のドットを表示
出力　なし
レジスタ　AF

REDCLK (01F5H)

機能　クロック・データを読む
入力　Cにクロック RAM アドレス

```
〔00MMAAAA〕
   |   └──── アドレス(0～15)
   └──────── モード(0～3)
```

出力　Aに読み込んだデータ（下位4ビットのみ有効）
レジスタ　F

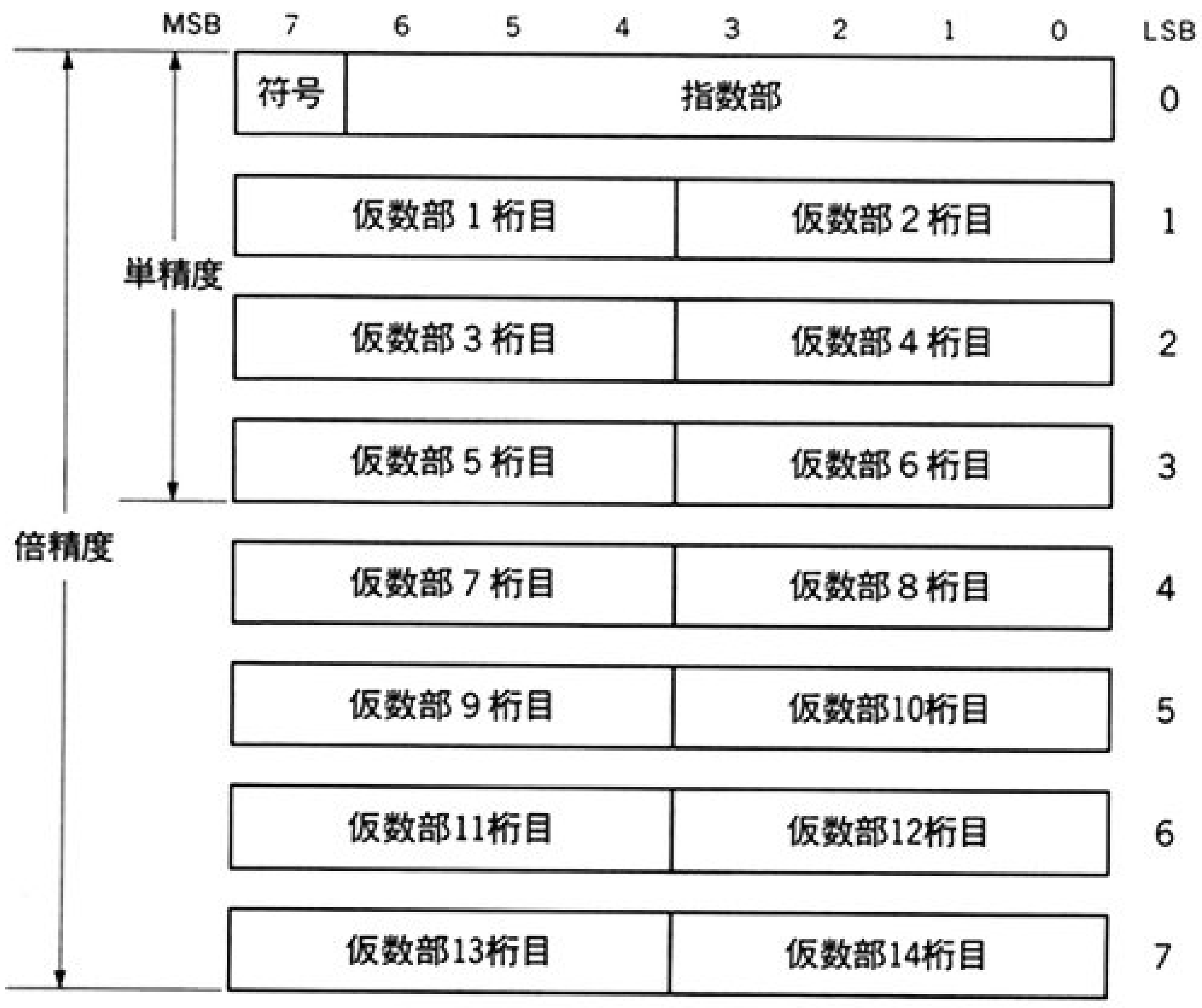
WRTCLK (01F9H)

機能　クロック・データを書き込む
入力　Aに書き込むデータ，Cにクロック RAM アドレス
出力　なし
レジスタ　F

A.2 Math-Pack

Math-Pack とは，MSX-BASIC の数値演算ルーチンの中核をなすもので，これらを機械語から呼び出すことにより，簡単に浮動小数点演算や三角関数ルーチンを利用することができます．

Math-Pack での実数演算はすべて BCD(Binary Coded Decimal＝2 進化 10 進法)表現によって行われます．また，実数表現には“単精度”と“倍精度”の 2 種類があり，単精度(6桁)は 4 バイト，倍精度(14 桁)は 8 バイトで表現されます(図 A.1，図 A.2)．



図A.1
BCDによる実数表現のフォーマット

単精度表現の例

123456 → 0.123456 E+6

	0	1	2	3
DAC	46	12	34	56

倍精度表現の例

123456.78901234 → 0.12345678901234 E+6

	0	1	2	3	4	5	6	7
DAC	46	12	34	56	78	90	12	34

図A.2 実数の表現例

1 つの実数は，符号部，指数部，仮数部から成り立っています．符号部とはこの仮数の符号を表し，0 ならば正，1 ならば負となります．指数部はバイナリ表現であり，+63 乗～−63 乗まで表すことができます(図 A.3)．倍精度実数の有効範囲を図 A.4 に示します．

符号	指数部							意味
0	0	0	0	0	0	0	0	………0
1	0	0	0	0	0	0	0	………未定義(−0?)
×	0	0	0	0	0	0	1	………10の−63乗
×	1	0	0	0	0	0	0	………10の0乗
×	1	1	1	1	1	1	1	………10の+63乗

注) "×"は，その値が1または0のどちらであっても関係がないことを表します

図A.3　指数部フォーマット

	0	1	2	3	4	5	6	7	(バイト)
DAC	FF	99	99	99	99	99	99	99	−0.99999999999999 E+63
				⋮					⋮
	81	10	00	00	00	00	00	00	−0.10000000000000 E−63
	00	×	×	×	×	×	×	×	0
	01	10	00	00	00	00	00	00	+0.10000000000000 E−63
				⋮					⋮
	7F	99	99	99	99	99	99	99	+0.99999999999999 E+63

図A.4　倍数度実数表現の有効範囲

Math-Packでは，演算の対象となるメモリがあらかじめ決められています．このメモリエリアを"DAC(Decmal ACcumlator)(F7F6H)"，それとの演算対象となる数値を格納するエリアを"ARG(F847H)"といいます．たとえば乗算ルーチンの場合，DACに入っている数とARGに入っている数の積が計算され，その結果がDACに入ることになります．

DACには，単精度実数，倍精度実数，2バイト整数を格納することができますが，これらのうちのどれが入っているのかを示すために"VALTYP(F663H)"が使われ，それぞれの場合について4，8，2の値が入ります．

単精度実数と倍精度実数は，いずれもDACの先頭から数値を格納しなければなりません．ただし，2バイト整数の場合だけは，下位，上位をDAC +2, +3に格納してください．

Math-PackはBASICの内部ルーチンであるため，エラーが起きた場合(0除算やオーバーフローなど)はそれに応じたエラールーチンへ自動的に分岐し，その後BASICコマンドレベルに戻ります．それを避けたいときは，H. ERRO(FFB1H)を書き換えてください．

● Math-Pack　ワークエリア

ラベル	アドレス	サイズ	意　味
VALTYP	F663H	1	DACに入っている数の形式を表す
DAC	F7F6H	16	BCD形式の浮動小数点アキュムレータ
ARG	F847H	16	DACの演算対象

● Math-Pack　エントリ

基本演算

ラベル	アドレス	機　能
DECSUB	268CH	DAC ← DAC－ARG
DECADD	269AH	DAC ← DAC＋ARG
DECNRM	26FAH	DACを正規化する（※1）
DECROU	273CH	DACを四捨五入する
DECMUL	27E6H	DAC ← DAC＊ARG
DECDIV	289FH	DAC ← DAC／ARG

注）これらは，いずれもDAC，ARGにある数を倍精度実数として扱う
レジスタは，いずれも保存されない
※1 仮数部の不要な0を取り去ること（0.00123 → 0.123E-2）

関数1

ラベル	アドレス	機　能	変化するレジスタ
COS	2993H	DAC ← COS(DAC)	すべて
SIN	29ACH	DAC ← SIN(DAC)	すべて
TAN	29FBH	DAC ← TAN(DAC)	すべて
ATN	2A14H	DAC ← ATN(DAC)	すべて
LOG	2A72H	DAC ← LOG(DAC)	すべて
SQR	2AFFH	DAC ← SQR(DAC)	すべて
EXP	2B4AH	DAC ← EXP(DAC)	すべて
RND	2BDFH	DAC ← RND(DAC)	すべて

注）すべてBASICにおける同名の関数の処理ルーチンである
変化するレジスタすべてとは，A, B, C, D, E, H, Lの各レジスタを指す

関数2

ラベル	アドレス	機　能	変化するレジスタ
SIGN	2E71H	A ← DACの符号	A
ABSFN	2E82H	DAC ← ABS(DAC)	すべて
NEG	2E8DH	DAC ← NEG(DAC)	A, H, L
SGN	2E97H	DAC ← SGN(DAC)	A, H, L

注）エントリラベルSIGN以外のルーチンは，すべてBASICにおける同名の関数の処理ルーチンである
変化するレジスタすべてとは，A, B, C, D, E, H, Lの名レジスタを指す
SGNは，結果が2バイト整数で表されるので注意

移動

ラベル	アドレス	機　能	対　象	変化するレジスタ
MAF	2C4DH	ARG ← DAC	倍精度	A, B, D, E, H, L
MAM	2C50H	ARG ← 〔HL〕	倍精度	A, B, D, E, H, L
MOV8DH	2C53H	〔DE〕 ← 〔HL〕	倍精度	A, B, D, E, H, L
MFA	2C59H	DAC ← ARG	倍精度	A, B, D, E, H, L
MFM	2C5CH	DAC ← 〔HL〕	倍精度	A, B, D, E, H, L
MMF	2C67H	〔HL〕 ← DAC	倍精度	A, B, D, E, H, L
MOV8HD	2C6AH	〔HL〕 ← 〔DE〕	倍精度	A, B, D, E, H, L
XTF	2C6FH	〔SP〕 ⟷ DAC	倍精度	A, B, D, E, H, L
PHA	2CC7H	ARG → 〔SP〕	倍精度	A, B, D, E, H, L
PHF	2CCCH	DAC → 〔SP〕	倍精度	A, B, D, E, H, L
PPA	2CDCH	〔SP〕 → ARG	倍精度	A, B, D, E, H, L
PPF	2CE1H	〔SP〕 → DAC	倍精度	A, B, D, E, H, L
PUSHF	2EB1H	DAC → 〔SP〕	単精度	D, E
MOVFM	2EBEH	DAC ← 〔HL〕	単精度	B, C, D, E, H, L
MOVFR	2EC1H	DAC ← (CBED)	単精度	D, E
MOVRF	2ECCH	(CBED) ← DAC	単精度	B, C, D, E, H, L
MOVRMI	2ED6H	(CBED) ← 〔HL〕	単精度	B, C, D, E, H, L
MOVRM	2EDFH	(BCDE) ← 〔HL〕	単精度	B, C, D, E, H, L
MOVMF	2EE8H	〔HL〕 ← DAC	単精度	A, B, D, E, H, L
MOVE	2EEBH	〔HL〕 ← 〔DE〕	単精度	B, C, D, E, H, L
VMOVAM	2EEFH	ARG ← 〔HL〕	VALTYP	B, C, D, E, H, L
MOVVFM	2EF2H	〔DE〕 ← 〔HL〕	VALTYP	B, C, D, E, H, L
VMOVE	2EF3H	〔HL〕 ← 〔DE〕	VALTYP	B, C, D, E, H, L
VMOVFA	2F05H	DAC ← ARG	VALTYP	B, C, D, E, H, L
VMOVFM	2F08H	DAC ←〔HL〕	VALTYP	B, C, D, E, H, L
VMOVAF	2F0DH	ARG ← DAC	VALTYP	B, C, D, E, H, L
VMOVMF	2F10H	〔HL〕 ← DAC	VALTYP	B, C, D, E, H, L

注）〔HL〕, 〔DE〕は，それぞれHL, DEレジスタが指したメモリ上の数値を示す
　（　）内の4レジスタ名は，左から（符号＋指数部），（仮数部1，2桁），（仮数部3，4桁），（仮数部5，6桁）を示す単精度実数である
　移動の対象がVALTYPとなっているものは，VALTYP（F663H番地）に示されている型に応じた移動（2, 4, 8バイト）を行う

比較

ラベル	アドレス	対　象	左辺	右辺	変化するレジスタ
FCOMP	2F21H	単精度実数	CBED	DAC	HL
ICOMP	2F4DH	2バイト整数	DE	HL	HL
XDCOMP	2F5CH	倍精度実数	ARG	DAC	すべて

注）結果はすべてAレジスタに入る．レジスタの意味は，
　A＝1　→ 左辺＜右辺
　A＝0　→ 左辺＝右辺
　A＝－1 → 左辺＞右辺
　単精度の比較で，左辺のCBEDとは各レジスタに，それぞれ単精度の（符号＋指数部），（仮数部1，2桁），（仮数部3，4桁），（仮数部5，6桁）が入ることを意味する

浮動小数点入出力

ラベル	アドレス	機　　能
FIN	3299H	浮動小数点を表す文字列を，実数表現に変換してDACに格納する

エントリー条件　HL　←　文字列の先頭アドレス
　　　　　　　　A　←　文字列の先頭文字
リターン条件　DAC ←　実数
　　　　　　　　C　←　FFH：小数点なし，0：小数点あり
　　　　　　　　B　←　小数点以降のケタ数
　　　　　　　　D　←　総ケタ数

ラベル	アドレス	機　　能
FOUT	3425H	DACに入っている実数表現の数を文字列に変換する（フォーマット指定なし）
PUFOUT	3426H	DACに入っている実数表現の数を文字列に変換する（フォーマット指定あり）

エントリー条件　A　←　変換のフォーマット
　　bit7 0：フォーマット指定なし，1：フォーマット指定あり
　　bit6 0：コンマによる区切りなし，1：3ごとにコンマで区切る
　　bit5 0：意味なし，1：先頭のスペースを"＊"でうめる
　　bit4 0：意味なし，1：数値の前に"$"を付ける
　　bit3 0：意味なし，1：正の数の場合も符号"＋"を付ける
　　bit2 0：意味なし，1：数値の後ろに符号を付ける
　　bit1 未使用
　　bit0 0：固定小数点，1：浮動小数点
　　B　←　小数点を含まない，小数点以前のケタ数
　　C　←　小数点を含む，小数点以降のケタ数
リターン条件　HL　←　文字列の先頭アドレス

ラベル	アドレス	機　　能
FOUTB	371AH	DAC＋2，3に入っている2バイト整数を2進表現の文字列に変換する
FOUTO	371EH	DAC＋2，3に入っている2バイト整数を8進表現の文字列に変換する
FOUTH	3722H	DAC＋2，3に入っている2バイト整数を16進表現の文字列に変換する

エントリー条件　DAC＋2 ← 2バイト整数
　　　　　　　　VALTYP ← 2
リターン条件　HL ←文字列の先頭アドレス

注）レジスタはいずれも保存されない
　出力ルーチンにおける文字列先頭アドレスは，通常FBUFFR(F7C5H番地から)内となるが，場合により先頭アドレスは前後する
　FOUTB，FOUTO，FOUTHに限らずDAC＋2に入れた整数を扱うときには，VALTYP(F663H番地)に必ず2を入れること

型変換

ラベル	アドレス	機能
FRCINT	2F8AH	DACを2バイト整数にする(DAC+2, 3)
FRCSNG	2FB2H	DACを単精度実数にする
FRCDBL	303AH	DACを倍精度実数にする
FIXER	30BEH	DAC ← SGN(DAC) * INT(ABS(DAC))

注) 実行後VALTYP(F663H番地)には, DACの型を表す数(2, 4, 8)が入る
レジスタは, いずれも保存されない

整数演算

ラベル	アドレス	機能	変化するレジスタ
UMULT	314AH	DE ← BC*DE	A, B, C, D, E
ISUB	3167H	HL ← DE−HL	すべて
IADD	3172H	HL ← DE+HL	すべて
IMULT	3193H	HL ← DE*HL	すべて
IDIV	31E6H	HL ← DE/HL	すべて
IMOD	323AH	HL ← DE mod HL(DE ← DE/HL)	すべて

べき乗

ラベル	アドレス	機能	基数	指数	結果
SNGEXP	37C8H	単精度実数のべき乗を求める	DAC	ARG	DAC
DBLEXP	37D7H	倍精度実数のべき乗を求める	DAC	ARG	DAC
INTEXP	383FH	2バイト整数のべき乗を求める	DE	HL	DAC

注) レジスタは, いずれも保存されない

A.3 ビットブロック・トランスファ

ビットブロック・トランスファはBASICのCOPY命令に相当し，RAM，VRAM，ディスク間でデータを転送するものです．これは拡張ROM内のルーチンによって実行され，マシン語からも簡単に使うことができます．拡張ROM内にあるため，このルーチンを利用する場合は，BIOSのSUBROMあるいはEXTROMを使ってください．

1. VRAM間の転送

● BLTVV(0191H/SUB)

機　能　VRAM領域内で転送を行う

入　力　HLレジスタ←F562H

さらに以下のパラメータを設定する

・SX(F562H, 2)	；ソースのX座標
・SY(F564H, 2)	；ソースのY座標
・DX(F566H, 2)	；デスティネーションのX座標
・DY(F568H, 2)	；デスティネーションのY座標
・NX(F56AH, 2)	；X方向のドット数
・NY(F56CH, 2)	；Y方向のドット数
・CDUMMY(F56EH, 1)	；ダミー(セットする必要はない)
・ARG(F56FH, 1)	；方向と拡張RAMの選択(VDPのR#45と同じ)
・LOGOP(F570H, 1)	；ロジカルオペレーション・コード(VDPのロジカルオペレーション・コードと同じ)

出　力　CYフラグをリセット

レジスタ　すべて

2. RAMとVRAM間のデータ転送

以下のルーチンを使う場合，グラフィックのデータ領域として，スクリーンモードによって次のような大きさの領域を確保しておく必要があります．

・スクリーンモード6

X方向のドット数*Y方向のドット数/4+4

・スクリーンモード5または7

X方向のドット数*Y方向のドット数/2+4

・スクリーンモード8

X方向のドット数＊Y方向のドット数/2+4

ただし，割り算の端数を切り上げてください．なお，ディスクやRAMでは配列データと同様にサイズを示すデータが付加されます．このデータは先頭から2バイトがX方向のドット数，次の2バイトがY方向のドット数を示しています．

● BLTVM(0195H/SUB)

機　能　配列をVRAMに転送する

入　力　HLレジスタ← F562H

さらに以下のパラメータを設定する

・DPTR(F562H, 2)	；メモリのソース・アドレス
・DUMMY(F564H, 2)	；ダミー(セットする必要はない)
・DX(F566H, 2)	；デスティネーションのX座標
・DY(F568H, 2)	；デスティネーションのY座標
・NX(F56AH, 2)	；X方向のドット数(セットする必要はない．これは転送するデータの先頭にすでに持っている)
・NY(F56CH, 2)	；Y方向のドット数(セットする必要はない．これは転送するデータの先頭にすでに持っている)
・CDUMMY(F56EH, 1)	；ダミー(セットする必要はない)
・ARG(F56FH, 1)	；方向と拡張RAMの選択(VDPのR#45と同じ)
・LOGOP(F570H, 1)	；ロジカル・オペレーション・コード(VDPのロジカル・オペレーション・コードと同じ)

出　力　転送するデータの個数がおかしければCYフラグをセット

レジスタ　すべて

● BLTMV(0199H/SUB)

機　能　VRAMから配列に転送する

入　力　HLレジスタ← F562H

さらに以下のパラメータを設定する

・SX(F562H, 2) ；ソースのX座標
・SY(F564H, 2) ；ソースのY座標
・DPTR(F566H, 2) ；メモリのデスティネーション・アドレス
・DUMMY(F568H, 2) ；ダミー(セットする必要はない)
・NX(F56AH, 2) ；X方向のドット数
・NY(F56CH, 2) ；Y方向のドット数
・CDUMMY(F56EH, 1) ；ダミー(セットする必要はない)
・ARG(F56FH, 1) ；方向と拡張RAMの選択(VDPのR#45と同じ)

出　力　CYフラグをリセット

レジスタ　すべて

3. ディスクとRAM, VRAM間の転送

ディスクを使う場合, まずファイル名を設定する必要があります(ファイル名はBASICと同様に指定). 例を次に示します.

```
          ⋮
        LD    HL, FNAME                 ;  Get pointer to file name
        LD    (FNPTR), HL               ;  Set it to parameter area
          ⋮
FNAME:  DB    22H, 'B : TEST. PIC', 22H, 0   ;  "TEST. PIC", end mark
```

ここで, 何かエラーが起きれば, BASICインタープリタのエラーハンドラへジャンプします. そのため, 自分のプログラム内でエラーをハンドリングしたい場合やMSX-DOS, ROMカートリッジでこのルーチンを呼ぶ場合, フックをセットする必要があります. このフックはH.ERRO(FFB1H)です.

● BLTVD(019DH/SUB)

機　能　ディスクからVRAMへ転送する

入　力　HLレジスタ←F562H

さらに以下のパラメータを設定する

・FNPTR(F562H, 2) ；ファイル名のあるアドレス
・DUMMY(F564H, 2) ；ダミー(セットする必要はない)
・DX(F566H, 2) ；デスティネーションのX座標
・DY(F568H, 2) ；デスティネーションのY座標

・NX(F56AH, 2) ；X方向のドット数(セットする必要はない．これは転送するデータの先頭にすでに持っている)
・NY(F56CH, 2) ；Y方向のドット数(セットする必要はない．これは転送するデータの先頭にすでに持っている)
・CDUMMY(F56EH, 1) ；ダミー(セットする必要はない)
・ARG(F56FH, 1) ；方向と拡張RAMの選択(VDPのR#45と同じ)
・LOGOP(F570H, 1) ；ロジカル・オペレーション・コード(VDPのロジカル・オペレーション・コードと同じ)

出　力　パラメータにエラーがあればCYフラグをセット
レジスタ　すべて

● BLTDV(01A1H/SUB)

機　能　VRAMからディスクへ転送する
入　力　HLレジスタ← F562H
さらに以下のパラメータを設定する
・SX(F562H, 2) ；ソースのX座標
・SY(F564H, 2) ；ソースのY座標
・FNPTR(F566H, 2) ；ファイル名のあるアドレス
・DUMMY(F568H, 2) ；ダミー(セットする必要はない)
・NX(F56AH, 2) ；X方向のドット数
・NY(F56CH, 2) ；Y方向のドット数
・CDUMMY(F56EH, 1) ；ダミー(セットする必要はない)
・ARG(F56FH, 1) ；方向と拡張RAMの選択(VDPのR#45と同じ)

出　力　CYフラグをリセット
レジスタ　すべて

● BLTMD(01A5H/SUB)

機　能　ディスクから配列データをロードする
入　力　HLレジスタ← F562H
さらに以下のパラメータを設定する

・FNPTR(F562H, 2) ；ファイル名のあるアドレス
・SY(F564H, 2) ；ダミー(セットする必要はない)
・SPTR(F566H, 2) ；ロードする先頭アドレス
・EPTR(F568H, 2) ；ロードする最終アドレス

出 力 CY フラグをリセット

レジスタ すべて

● BLTDM(01A9H/SUB)

機 能 ディスクへ配列データをセーブする

入 力 HL レジスタ← F562H

さらに以下のパラメータを設定する

・SPTR(F562H, 2) ；セーブする先頭アドレス
・EPTR(F564H, 2) ；セーブする最終アドレス
・FNPTR(F566H, 2) ；ファイル名のあるアドレス

出 力 CY フラグをリセット

レジスタ すべて

A.4 ワークエリア一覧

MSX 2のワーク・エリアの概略を図A．5に示します．以下の説明では，この図のF380H～FFCAHのシステム・ワーク・エリアとフックについて解説します．表記は以下のとおりです．ただし，長さはバイト数です．

ラベル名(アドレス，長さ)

初期値，内容，使用目的



図A.5 ワークエリア

●インタースロットのリード，ライトコール用サブルーチン

RDPRIM(F380H, 5)

内容　基本スロットからの読み込み

WRPRIM(F385H, 7)

内容　基本スロットへ書き込み

CLPRIM(F38CH, 14)

内容　基本スロットコール

●USR関数のマシン語プログラムの開始アドレス，テキスト画面

USRTAB(F39AH, 20)

初期値　FCERR

内容　USR関数のマシン語プログラム(0～9)の開始番地，機械語プログラム定義前の値はすべてエラールーチンFCERR(475AH)を指す

LINL40(F3AEH, 1)

初期値　39

内容　SCREEN0のときの1行の幅(SCREEN0のときのWIDTH文により設定される)

LINL32(F3AFH, 1)
　初期値　29
　内容　SCREEN1のときの1行の幅(SCREEN1のときのWIDTH文により設定される)

LINLEN(F3B0H, 1)
　初期値　29
　内容　現在の画面の1行の幅

CRTCNT(F3B1H, 1)
　初期値　24
　内容　現在の画面の行数

CLMLST(F3B2H, 1)
　初期値　14
　内容　PRINT命令において各項目がカンマで区切られている場合の横位置

●初期化用ワーク

・SCREEN 0

TXTNAM(F3B3H, 2)
　初期値　0000H
　内容　パターンネーム・テーブル

TXTCOL(F3B5H, 2)
　内容　使用せず

TXTCGP(F3B7H, 2)
　初期値　0800H
　内容　パターンジェネレータ・テーブル

TXTATR(F3B9H, 2)
　内容　使用せず

TXTPAT(F3BBH, 2)
　内容　使用せず

・SCREEN 1

T32NAM(F3BDH, 2)
　初期値　1800H
　内容　パターンネーム・テーブル

T32COL(F3BFH, 2)
　初期値　2000H
　内容　カラーテーブル

T32CGP(F3C1H, 2)
　初期値　0000H
　内容　パターンジェネレータ・テーブル

T32ATR(F3C3H, 2)
　初期値　1B00H
　内容　スプライトアトリビュート・テーブル

T32PAT(F3C5H, 2)
　初期値　3800H
　内容　スプライト・ジェネレータ・テーブル

・SCREEN 2

GRPNAM(F3C7H, 2)
　初期値　1800H
　内容　パターンネーム・テーブル

GRPCOL(F3C9H, 2)
　初期値　2000H
　内容　カラーテーブル

GRPCGP(F3CBH, 2)
　初期値　0000H
　内容　パターンジェネレータ・テーブル

GRPATR(F3CDH, 2)
　初期値　1B00H
　内容　スプライトアトリビュート・テーブル

GRPPAT(F3CFH, 2)
　初期値　3800H
　内容　スプライト・ジェネレータ・テーブル

・SCREEN 3

MLTNAM(F3D1H, 2)
　初期値　0800H
　内容　パターンネーム・テーブル

MLTCOL(F3D3H, 2)
　内容　使用せず

MLTCGP(F3D5H, 2)
　初期値　0000H
　内容　パターンジェネレータ・テーブル

MLTATR(F3D7H, 2)
　初期値　1B00H
　内容　スプライトアトリビュート・テーブル

MLTPAT(F3D9H, 2)
　初期値　3800H
　内容　スプライト・ジェネレータ・テーブル

●その他のスクリーン設定

CLIKSW(F3DBH, 1)
　初期値　1
　内容　キークリックスイッチ(0=OFF, 0以外=ON). SCREEN文の<キークリックスイッチ>により設定される

CSRY(F3DCH, 1)
　初期値　1
　内容　カーソルのY座標

CSRX(F3DDH, 1)
　初期値　1
　内容　カーソルのX座標

CNSDFG(F3DEH, 1)
　初期値　0

内容　ファンクションキー表示スイッチ(0=表示あり，0以外=表示なし)．KEY ON／OFF文によって設定される

●VDPレジスタのセーブエリアなど

RG0SAV(F3DFH, 1)
　初期値　0
RG1SAV(F3E0H, 1)
　初期値　E0H
RG2SAV(F3E1H, 1)
　初期値　0
RG3SAV(F3E2H, 1)
　初期値　0
RG4SAV(F3E3H, 1)
　初期値　0
RG5SAV(F3E4H, 1)
　初期値　0
RG6SAV(F3E5H, 1)
　初期値　0
RG7SAV(F3E6H, 1)
　初期値　0
STATFL(F3E7H, 1)
　初期値　0
　内容　VDPのステータスを保存(MSX2ではステータスレジスタ0の内容)
TRGFLG(F3E8H, 1)
　初期値　FFH
　内容　ジョイスティックのトリガボタンの状態を保存する
FORCLR(F3E9H, 1)
　初期値　15
　内容　前景色．COLOR文で設定される
BAKCLR(F3EAH, 1)
　初期値　4
　内容　背景色．COLOR文で設定される
BDRCLR(F3EBH, 1)
　初期値　7
　内容　周辺色．COLOR文で設定される
MAXUPD(F3ECH, 3)
　初期値　JP　0000H　(C3H, 00H, 00H)
　内容　CIRCLE文が内部で使用
MINUPD(F3EFH, 3)
　初期値　JP　0000H　(C3H, 00H, 00H)
　内容　CIRCLE文が内部で使用
ATRBYT(F3F2H, 1)
　初期値　15
　内容　グラフィック使用時のカラーコード

●PLAY文用ワークエリア

QUEUES(F3F3H, 2)
　初期値　QUETAB(F959H)
　内容　PLAY文実行時のキューテーブルを指す
FRCNEW(F3F5H, 1)
　初期値　255
　内容　BASICインタープリタが内部で使用する

●キー入力用ワークエリア

SCNCNT(F3F6H, 1)
　初期値　1
　内容　キースキャンの時間間隔
REPCNT(F3F7H, 1)
　初期値　50
　内容　キーのオートリピートが開始するまでの時間
PUTPNT(F3F8H, 2)
　初期値　KEYBUF(FBF0H)
　内容　キーバッファへの書き込みを行う番地を指す
GETPNT(F3FAH, 2)
　初期値　KEYBUF(FBF0H)
　内容　キーバッファからの読み込みを行う番地を指す

●カセット用パラメータ

CS120(F3FCH, 5*2)
・1200ボー
　内容(値)　83(LOW01)……………ビット0を表すLOWの幅
　92(HIGH01)……………ビット0を表すHIGHの幅
　38(LOW11)……………ビット1を表すLOWの幅
　45(HIGH11)……………ビット1を表すHIGHの幅
　HEADLEN*2／256…ショートヘッダ用のヘッダビットのHIGHバイト(HEDLEN=2000)
・2400ボー
　内容(値)　37(LOW02)……………ビット0を表すLOWの幅

45(HIGH02)……………ビット0を表すHIGHの幅
14(LOW12) ……………ビット1を表すLOWの幅
22(HIGH12)……………ビット1を表すHIGHの幅
HEADLEN＊4／256 …ショートヘッダ用のヘッダビットのHighバイト(HEDLEN＝2000)

LOW(F406H, 2)
初期値　LOW01, HIGH01(デフォルト1200ボー)
内容　現在のボーレートのビット0を表すLOWとHIGHの幅. SCREEN文の<カセットボーレート>により設定される

HIGH(F408H, 2)
初期値　LOW11, HIGH11(デフォルト1200ボー)
内容　現在のボーレートのビット1を表すLOWとHIGHの幅. SCREEN文の<カセットボーレート>により設定される

HEADER(F40AH, 1)
初期値　HEADLEN＊2/256(デフォルト1200ボー)
内容　現在のボーレートのショートヘッダ用のヘッダビット(HEDLEN＝2000). SCREEN文の<カセットボーレート>により設定される

ASPCT1(F40BH, 2)
内容　256/アスペクト比. CIRCLE文で使用するためにSCREEN文で設定される

ASPCT2(F40DH, 2)
内容　256＊アスペクト比. CIRCLE文で使用するためにSCREEN文で設定される

ENDPRG(F40FH, 5)
初期値　“：”
内容　RESUME NEXT文のための仮のプログラムの終わり

● BASICが内部で使うワーク

ERRFLG(F414H, 1)
内容　エラー番号を保存するためのエリア

LPTPOS(F415H, 1)
初期値　0
内容　プリンタのヘッド位置

PRTFLG(F416H, 1)
内容　プリンタへ出力するかどうかのフラグ

NTMSXP(F417H, 1)
内容　プリンタ種別(0＝MSX用プリンタ, 0以外＝MSX用プリンタでない)

RAWPRT(F418H, 1)
内容　raw-modeでプリント中なら0以外

VLZADR(F419H, 2)
内容　VAL関数で置き換えられる文字のアドレス

VLZDAT(F41BH, 1)
内容　VAL関数で0に置き換わる文字

CURLIN(F41CH, 2)
内容　BASICが現在実行中の行番号

KBUF(F41FH, 318)
内容　クランチバッファ. BUF(F55EH)から中間言語に直されて入る

BUFMIN(F55DH, 1)
初期値　“，”
内容　INPUT文で使われる

BUF(F55EH, 258)
内容　タイプした文字が入るバッファ. ダイレクトステートメントがアスキーコードで入る

ENDBUF(F660H, 1)
内容　BUF(F55EH)がオーバーフローするのを防ぐ

TTYPOS(F661H, 1)
内容　BASICが内部で持つ仮想的なカーソル位置

DIMFLG(F662H, 1)
内容　BASICが内部で使用する

VALTYP(F663H, 1)
内容　変数の型の識別に使用する

DORES(F664H, 1)
内容　保存されている語がクランチできるかどうかを示す

DONUM(F665H, 1)
内容　クランチ用のフラグ

CONTXT(F666H, 2)
内容　CHRGETで使うテキストアドレスの保存

CONSAV(F668H, 1)
内容　CHRGETが呼ばれた後の定数のトーク

ンを保存

CONTYP(F669H, 1)

内容　保存した定数のタイプ

CONLO(F66AH, 8)

内容　保存した定数の値

MEMSIZ(F672H, 2)

内容　BASIC が使用するメモリの最上位番地

STKTOP(F674H, 2)

内容　BASIC がスタックとして使用する番地. CLEAR 文により変化する

TXTTAB(F676H, 2)

内容　BASIC テキストエリアの先頭番地

TEMPPT(F678H, 2)

初期値　TEMPST(F67AH)

内容　テンポラリディスクリプタの空きエリアの先頭番地

TEMPST(F67AH, 3 * NUMTMP)

内容　NUMTEMP 用の領域

DSCTMP(F698H, 3)

内容　ストリング関数の答えのストリングディスクリプタが入る

FRETOP(F69BH, 2)

内容　文字列領域の空きエリアの先頭番地

TEMP3(F69DH, 2)

内容　ガベージコレクションや USR 関数などに使われる

TEMP8(F69FH, 2)

内容　ガベージコレクション用

ENDFOR(F6A1H, 2)

内容　FOR 文の次の番地を保存する(ループ時に FOR 文の次から実行するため)

DATLIN(F6A3H, 2)

内容　READ 文の実行により読まれた DATA 文の行番号

SUBFLG(F6A5H, 1)

内容　USR 関数などで配列を使うときのフラグ

FLGINP(F6A6H, 1)

内容　INPUT や READ で使われるフラグ

TEMP(F6A7H, 2)

内容　ステートメントコードのための一時保存場所, 変数ポインタ, テキストアドレスなどに使用する

PTRFLG(F6A9H, 1)

内容　変換する行番号がなければ 0, あれば 0 以外

AUTFLG(F6AAH, 1)

内容　AUTO コマンド有効, 無効フラグ(0以外=有効中, 0=無効中)

AUTLIN(F6ABH, 2)

内容　一番新しく入力された行番号

AUTINC(F6ADH, 2)

初期値　10

内容　AUTO コマンドの行番号の増分値

SAVTXT(F6AFH, 2)

内容　実行中のテキストのアドレスを保存する領域. 主に RESUME 文によるエラー回復で使用される

SAVSTK(F6B1H, 2)

内容　スタックを保存する領域. 主にエラーが起きたとき, エラー回復ルーチンがスタックをリストアするために使用される

ERRLIN(F6B3H, 2)

内容　エラーが起きたときの行番号

DOT(F6B5H, 2)

内容　何らかの形で画面に表示された, あるいは入力された最新の行番号

ERRTXT(F6B7H, 2)

内容　エラーが起きたテキストのアドレス. 主に RESUME 文によるエラー回復で使用される

ONELIN(F6B9H, 2)

内容　エラーが起きたときの飛び先行のテキストアドレス. ON ERROR GOTO 文により設定される

ONEFLG(F6BBH, 1)

内容　エラールーチンの実行中を示すフラグ. (0以外=実行中, 0=実行中でない)

TEMP2(F6BCH, 2)

内容　一時保存用

OLDLIN(F6BEH, 2)

内容　Ctrl+STOP, STOP 命令, END 命令で中断されたか, あるいは最後に実行された行番号

OLDTXT(F6C0H, 2)

内容　次に実行する文のテキストアドレス

VARTAB(F6C2H, 2)

内容　単純変数の開始番地. NEW 文を実行すると〔TXTTAB(F676H)の内容+2〕が設定される

ARYTAB(F6C4H, 2)

内容　配列テーブルの開始番地

STREND(F6C6H, 2)

内容 テキストエリアや変数エリアとして使用中であるメモリの最後の番地

DATPTR(F6C8H, 2)

内容 READ 文の実行により読まれたデータのテキストアドレス

DEFTBL(F6CAH, 26)

内容 英文字1字に対し変数の型を保持するエリア. CLEAR, DEFSTR, !, #などの型宣言で変化する

●ユーザー関数のパラメータに関するワーク

PRMSTK(F6E4H, 2)

内容 スタック上の以前の定義ブロック(ガベージコレクション用)

PRMLEN(F6E6H, 2)

内容 処理対象のテーブルのバイト数

PARM1(F6E8H, PRMSIZ)

内容 処理対象のパラメータ定義テーブル. PRMSIZ は定義ブロックのバイト数で初期値は 100

PRMPRV(F74CH, 2)

初期値 PRMSTK

内容 以前のパラメータブロックのポインタ(ガベージコレクション用)

PRMLN2(F74EH, 2)

内容 パラメータブロックの大きさ

PARM2(F750H, 100)

内容 パラメータの保存用

PRMFLG(F7B4H, 1)

内容 PARM1 がサーチ済みかどうかを示すフラグ

ARYTA2(F7B5H, 2)

内容 サーチの終点

NOFUNS(F7B7H, 1)

内容 処理対象関数がない場合は 0

TEMP9(F7B8H, 2)

内容 ガベージコレクション用の一時保存場所

FUNACT(F7BAH, 2)

内容 処理対象関数の数

SWPTMP(F7BCH, 8)

内容 SWAP 文の最初の変数の値の一時保存場所

TRCFLG(F7C4H, 1)

内容 トレースフラグ. (0以外=TRACE ON, 0=TRACE OFF)

● Math-Pack 用ワーク

FBUFFR(F7C5H, 43)

内容 マスパックが内部で使用する

DECTMP(F7F0H, 2)

内容 10 進整数を浮動小数点数にするときに使用する

DECTM2(F7F2H, 2)

内容 除算ルーチンの実行時に使用する

DECCNT(F7F4H, 1)

内容 除算ルーチンの実行時に使用する

DAC(F7F6H, 16)

内容 演算の対象となる値を設定するエリア

HOLD8(F806H, 48)

内容 10 進数の乗算のためのレジスタ保存エリア

HOLD2(F836H, 8)

内容 マスパックが内部で使用する

HOLD(F83EH, 8)

内容 マスパックが内部で使用する

ARG(F847H, 16)

内容 DAC(F7F6H)との演算対象となる値を設定するエリア

RNDX(F857H, 8)

内容 最新の乱数を倍精度実数で保存する. RND 関数で設定される

● BASIC インタープリタが使うデータエリア

MAXFIL(F85FH, 1)

内容 ファイル番号の最大値. MAXFILES 文により設定される

FILTAB(F860H, 2)

内容 ファイルデータエリアの先頭番地

NULBUF(F862H, 2)

内容 SAVE, LOAD で BASIC インタープリタが使用するバッファ

PTRFIL(F864H, 2)

内容 アクセス中のファイルのファイルデータがある番地

RUNFLG(F866H, 0)

内容 プログラムをロード後実行するなら 0 でない値. LOAD 文の R オプションなどで使用する

FILNAM(F866H, 11)

内容 ファイル名の保存エリア

FILNM2(F871H, 11)

内容 ファイル名の保存エリア

NLONLY(F87CH, 1)
　内容　プログラムロード中は0でない値となる
SAVEND(F87DH, 2)
　内容　セーブするマシン語プログラムの最終番地
FNKSTR(F87FH, 160)
　内容　ファンクションキーの文字列保存エリア(16文字×10)
CGPNT(F91FH, 3)
　内容　ROM上の文字フォント格納アドレス
NAMBAS(F922H, 2)
　内容　現在のパターンネーム・テーブルのベース番地
CGPBAS(F924H, 2)
　内容　現在のパターン・ジェネレーター・テーブルのベース番地
PATBAS(F926H, 2)
　内容　現在のスプライト・ジェネレーター・テーブルのベース番地
ATRBAS(F928H, 2)
　内容　現在のスプライトアトリビュート・テーブルのベース番地
CLOC(F92AH, 2)
　内容　グラフィックルーチンが内部で使用する
CMASK(F92CH, 1)
　内容　グラフィックルーチンが内部で使用する
MINDEL(F92DH, 2)
　内容　グラフィックルーチンが内部で使用する
MAXDEL(F92FH, 2)
　内容　グラフィックルーチンが内部で使用する

●CIRCLE文で使うデータエリア

ASPECT(F931H, 2)
　内容　円の縦横の比率. CLRCLE文の<比率>により設定される
CENCNT(F933H, 2)
　内容　CIRCLE文が内部で使用する
CLINEF(F935H, 1)
　内容　円の中心へ線を引くかどうかのフラグ. CIRCLE文の<角度>で指定
CNPNTS(F936H, 2)
　内容　プロットする点
CPLOTF(F938H, 1)
　内容　CIRCLE文が内部で使用する
CPCNT(F939H, 2)
　内容　円の1/8分割の数
CPCNT8(F93BH, 2)
　内容　CIRCLE文が内部で使用する
CRCSUM(F93DH, 2)
　内容　CIRCLE文が内部で使用する
CSTCNT(F93FH, 2)
　内容　CIRCLE文が内部で使用する
CSCLXY(F941H, 1)
　内容　xとyのスケール
CSAVEA(F942H, 2)
　内容　ADVGRPの保存エリア
CSAVEM(F944H, 1)
　内容　ADVGRPの保存エリア
CXOFF(F945H, 2)
　内容　中心からのxのオフセット
CYOFF(F947H, 2)
　内容　中心からのyのオフセット

●PAINT文で使用するデータエリア

LOHMSK(F949H, 1)
　内容　PAINT文が内部で使用する
LOHDIR(F94AH, 1)
　内容　PAINT文が内部で使用する
LOHADR(F94BH, 2)
　内容　PAINT文が内部で使用する
LOHCNT(F94DH, 2)
　内容　PAINT文が内部で使用する
SKPCNT(F94FH, 2)
　内容　スキップカウント
MIVCNT(F951H, 2)
　内容　移動カウント
PDIREC(F953H, 1)
　内容　ペイントの方向
LFPROG(F954H, 1)
　内容　PAINT文が内部で使用する
RTPROG(F955H, 1)
　内容　PAINT文が内部で使用する

●PLAYで使うデータエリア

MCLTAB(F956H, 2)
　内容　PLAYマクロ, あるいはDROWマクロのテーブルの先頭を指す
MCLFLG(F958H, 1)
　内容　PLAY／DRAWの指示
QUETAB(F959H, 24)
　内容　キューテーブル
　　+0　:　PUT　オフセット

＋1　：　GET　オフセット
＋2　：　バックアップ・キャラクタ
＋3　：　キューの長さ
＋4＋5：　キューのアドレス

QUEBAK(F971H, 4)
内容　BCKQ で使用する

VOICAQ(F975H, 128)
内容　音声 1 のキュー(1＝a)

VOICBQ(F9F5H, 128)
内容　音声 2 のキュー(2＝b)

VOICCQ(FA75H, 128)
内容　音声 3 のキュー(3＝c)

● MSX2 で追加されたワークエリア

DPPAGE(FAF5H, 1)
内容　ディスプレイページ番号

ACPAGE(FAF6H, 1)
内容　アクティブページ番号

AVCSAV(FAF7H, 1)
内容　AV コントロールポートの保存

EXBRSA(FAF8H, 1)
内容　SUB-ROM のスロットアドレス

CHRCNT(FAF9H, 1)
内容　バッファ中のキャラクタのカウンタ．ローマ字カナ変換で使用(値は 0<＝n<＝2)

ROMA(FAFAH, 2)
内容　バッファ中のキャラクタを入れておくエリア．ローマ字カナ変換で使用

MODE(FAFCH, 1)
内容　ローマ字カナ変換のモードスイッチと VRAM サイズ
〔K000WVVC〕

- C：1＝変換する，0＝変換しない
- VV：00＝16KVRAM　01＝64KVRAM　10＝128KVRAM
- W：0＝マスクする，1＝マスクしない
スクリーン 0～3 において VRAM アドレスを指定するときに 3FFFH と AND をとって設定するかどうかのフラグ，SCREEN4～8 ではつねにマスクしない
- K：1＝カタカナ，0＝ひらがな

NORUSE(FAFDH, 1)
内容　未使用

XSAVE(FAFEH, 2)
内容　〔I 0000000 XXXXXXXX〕

YSAVE(FB00H, 2)
内容　〔×0000000 YYYYYYYY〕
I＝1　ライトペンのインターラプト要求あり
0000000　＝符号なしオフセット
XXXXXXXX　＝　X座標
YYYYYYYY　＝　Y座標

LOGOPR(FB02H, 1)
内容　ロジカル・オペレーション・コード

● RS-232C で使うデータエリア

RSTMP(FB03H, 50)
内容　RS-232C またはディスクのワークエリア

TOCNT(FB03H, 1)
内容　RS-232C ルーチンが内部で使用する

RSFCB(FB04H, 2)
内容
FB04H＋0　：　RS-232C の LOW アドレス
FB04H＋1　：　RS-232C の HIGH アドレス

RSIQLN(FB06H, 1)
内容　RS-232C ルーチンが内部で使用する

MEXBIH(FB07H, 5)
内容
FB07H＋0　：　RST 30H(0F7H)
FB07H＋1　：　バイトデータ
FB07H＋2　：　(Low)
FB07H＋3　：　(Hogh)
FB07H＋4　：　RET　(0C9H)

OLDSTT(FB0CH, 5)
内容
FB0CH＋0　：　RST 30H(0F7H)
FB0CH＋1　：　バイトデータ
FB0CH＋2　：　(Low)

FB0CH+3 : (Hogh)
FB0CH+4 : RET (0C9H)

OLDINT(FB12H, 5)
内容 FB12H+0 : RST 30H(0F7H)
FB12H+1 : バイトデータ
FB12H+2 : (Low)
FB12H+3 : (Hogh)
FB12H+4 : RET (0C9H)

DEVNUM(FB17H, 1)
内容 RS232C ルーチンが内部で使用する

DATCNT(FB18H, 3)
内容 FB18H+0 : バイトデータ
FB18H+1 : バイトポインタ
FB18H+2 : バイトポインタ

ERRORS(FB1BH, 1)
内容 RS-232C ルーチンが内部で使用する

FLAGS(FB1BH, 1)
内容 RS-232C ルーチンが内部で使用する

ESTBLS(FB1DH, 1)
内容 RS-232C ルーチンが内部で使用する

COMMSK(FB1EH, 1)
内容 RS-232C ルーチンが内部で使用する

LSTCOM(FB1FH, 1)
内容 RS-232C ルーチンが内部で使用する

LSTMOD(FB20H, 1)
内容 RS-232C ルーチンが内部で使用する

● DOS が使用するデータエリア

リザーブ(FB21H～FB34H)
内容 DOS が使用する

● PLAY 文が使用するデータエリア (以下は MSX1 と共通)

PRSCNT(FB35H, 1)
内容 D1～D0 文字列パース
D7=0 1パス

SAVSP(FB36H, 2)
内容 プレー中のスタックポインタを保存

VOICEN(FB38H, 1)
内容 解釈中の現在の音声

SAVVOL(FB39H, 2)
内容 休止のために音量を保存する

MCLLEN(FB39H, 1)
内容 PLAY 文が内部で使用する

MCLPTR(FB3CH, 2)
内容 PLAY 文が内部で使用する

QUEUEN(FB3EH, 1)
内容 PLAY 文が内部で使用する

MUSICF(FC3FH, 1)
内容 音楽演奏用の割り込みフラグ

PLYCNT(FB40H, 1)
内容 キューに格納されている PLAY 文の数

● 音声スタティック・データエリアからの変位 (変位は 10 進数)

METREX(+0, 2)
内容 タイマカウントダウン

VCXLEN(+2, 1)
内容 この音声のための MCLLEN

VCXPTR(+3, 2)
内容 この音声のための MCLPTR

VCXSTP(+5, 2)
内容 スタックポインタの先頭を保存

QLENGX(+7, 1)
内容 キューに格納されるバイト数

NTICSX(+8, 2)
内容 新しいカウントダウン

TONPRX(+10, 2)
内容 トーンの周期を設定するエリア

AMPPRX(+12, 1)
内容 音量，エンベロープの区別

ENVPRX(+13, 2)
内容 エンベロープの周期を設定するエリア

OCTAVX(+15, 1)
内容 オクターブを設定するエリア

NOTELX(+16, 1)
内容 音の長さを設定するエリア

TEMPOX(+17, 1)
内容 テンポを設定するエリア

VOLUMX(+18, 1)
内容 音量を設定するエリア

ENVLPX(+19, 14)
内容 エンベロープの波形を設定するエリア

MCLSTX(+33, 3)
内容 スタックの保存場所

MCLSEX(+36, 1)
内容 初期化スタック

VCBSIZ(+37, 1)
内容 スタティックバッファの大きさ

●音声スタティック・データエリア

VCBA(FB41H, 37)
　内容　音声0のスタティックデータ

VCBB(FB66H, 37)
　内容　音声1のスタティックデータ

VCBC(FB8BH, 37)
　内容　音声2のスタティックデータ

●データエリア

ENSTOP(FBB0H, 1)
　内容　[SHIFT+Ctrl+GRAPH+かなキー]によるウォームスタートを可能にするフラグ(0=不可能, 0以外=可能)

BASROM(FBB1H, 1)
　内容　BASICテキストの存在場所を示す(0=RAM上, 0以外=ROM上)

LINTTB(FBB2H, 24)
　内容　ラインターミナルテーブル．テキスト画面の各行の情報を保持するエリア

FSTPOS(FBCAH, 2)
　内容　BIOSのINLIN(00B1H)で入力した行の最初の文字の位置

CODSAV(FBCCH, 1)
　内容　カーソルが重なった部分のキャラクタを保存するエリア

FNKSWI(FBCDH, 1)
　内容　KEY ON時にどのファンクションキーが表示されているか表す(1=F1～F5が表示, 0=F6～F10が表示)

FNKFLG(FBCEH, 10)
　内容　ON KEY GOSUB文により定義された行の実行を許可, 禁止, 停止するかファンクションキーごとに保存するためのエリア．KEY(n)ON／OFF／STOP文により設定される(0=KEY(n)OFF／STOP, 1=KEY(n)ON)

ONGSBF(FBD8H, 1)
　内容　TRPTBL(FC4CH)で待機中のイベントが発生したかどうかのフラグ

CLIKFL(FBD9H, 1)
　内容　キークリック・フラグ

OLDKEY(FBDAH, 11)
　内容　キーマトリクスの状態(旧)

NEWKEY(FBE5H, 11)
　内容　キーマトリクスの状態(新)

KEYBUF(FBF0H, 40)
　内容　キーコードバッファ

LINWRK(FC18H, 40)
　内容　スクリーンハンドラが使う一時保存場所

PATWRK(FC40H, 8)
　内容　パターンコンバータが使う一時保存場所

BOTTOM(FC48H, 2)
　内容　実装したRAMの先頭(低位)番地．MSX2では通常8000H

HIMEM(FC4AH, 2)
　内容　利用可能なメモリーの上位番地．CLEAR文の<メモリの上限>により設定される

TRPTBL(FC4CH, 78)
　内容　割り込み処理で使うトラップテーブル．ひとつのテーブルは3バイトで構成される1バイト目がON／OFF／STOP状態を表し，残りが分岐先のテキストアドレスを表す

FC4CH～FC69H(3*10バイト) ← ON KEY　GOSUBで使用
FC6AH～FC6CH(3*1バイト) ← ON STOP　GOSUBで使用
FC6DH～FC6FH(3*1バイト) ← ON SPRITE GOSUBで使用
FC70H～FC7EH(3*5バイト) ← ON STRIG GOSUBで使用
FC7FH～FC81H(3*1バイト) ← ON INTERVAL GOSUBで使用
FC82H～FC99H ← 拡張用

RTYCNT(FC9AH, 1)
　内容　BASICが内部で使用する

INTFLG(FC9BH, 1)
　内容　Ctrl+STOPが押された場合など，ここに03Hを入れることによりストップする

PADY(FC9CH, 1)
　内容　パドルのY座標

PADX(FC9DH, 1)
　内容　パドルのX座標

JIFFY(FC9EH, 2)
　内容　PLAY文が内部で使用する

INTVAL(FCA0H, 2)
　内容　インターバルの間隔．ON INTERVAL GOSUB文により設定される

INTCNT(FCA2H, 2)
　内容　インターバルのためのカウンタ

LOWLIM(FCA4H, 1)
内容 カセットテープからの読み込み中に使う

WINWID(FCA5H, 1)
内容 カセットテープからの読み込み中に使う

GRPHED(FCA6H, 1)
内容 グラフィックキャラクタを出す時のフラグ(1＝グラフィックキャラクタ，0＝通常の文字)

ESCCNT(FCA7H, 1)
内容 エスケープコードがきてから何文字目かをカウントするエリア

INSFLG(FCA8H, 1)
内容 挿入モードのフラグ(0＝通常モード，0以外＝挿入モード)

CSRSW(FCA9H, 1)
内容 カーソル表示の有無(0＝表示なし，0以外＝表示あり)
LOCATE 文の<カーソルスイッチ>により設定される

CSTYLE(FCAAH, 1)
内容 カーソルの形(0＝■，0以外＝_)

CAPST(FCABH, 1)
内容 CAPS キーの状態(0＝CAP OFF，0以外＝CAP ON)

KANAST(FCACH, 1)
内容 かなキーの状態(0＝かな OFF，0以外＝かな ON)

KANAMD(FCADH, 1)
内容 かなキー配列の状態(0＝50音配列，0以外＝JIS 配列)

FLBMEM(FCAEH, 1)
内容 BASIC プログラムをロード中は 0

SCRMOD(FCAFH, 1)
内容 現在のスクリーンモードの番号

OLDSCR(FCB0H, 1)
内容 スクリーンモード保存エリア

CASPRV(FCB1H, 1)
内容 CAS：が使う文字保存場所

BRDATR(FCB2H, 1)
内容 PAINT で使用する境界色のカラーコード．PAINT 文の<境界色>で指定される

GXPOS(FCB3H, 2)
内容 X座標

GYPOS(FCB5H, 2)
内容 Y座標

GRPACX(FCB7H, 2)
内容 グラフィックアキュムレータ(X座標)

GRPACY(FCB9H, 2)
内容 グラフィックアキュムレータ(Y座標)

DRWFLG(FCBBH, 1)
内容 DRAW 文で使用するフラグ

DRWSCL(FCBCH, 1)
内容 DRAW スケーリングファクタ(0＝スケーリングしない，0以外＝する)

DRWANG(FCBDH, 1)
内容 DRAW するときの角度

RUNBNF(FCBEH, 1)
内容 BLOAD 中，BSAVE 中，どちらでもない，のいずれかを表すフラグ

SAVENT(FCBFH, 2)
内容 BSAVE の開始番地

EXPTBL(FCC1H, 4)
内容 拡張スロット用のフラグテーブル．各スロットの拡張の有無

SLTTBL(FCC5H, 4)
内容 各拡張スロットレジスタ用の，現在のスロット選択状況

SLTATR(FCC9H, 64)
内容 各スロット用に属性を保存する

SLTWRK(FD09, 128)
内容 各スロット用に特定のワークエリアを確保する

PROCNM(FD89H, 16)
内容 拡張ステートメント(CALL 文の後)，拡張デバイス(OPEN の後)の名前が入る．0は終わり

DEVICE(FD99H, 1)
内容 カートリッジ用の装置識別に使用する

●フック

H.KEYI(FD9AH)
意味 MSXIO 割り込み処理の始め
使用目的 RS-232C などの割り込み処理を追加する

H.TIMI(FD9FH)
意味 MSXIO タイマ割り込み処理
使用目的 タイマー割り込み処理を追加するため

H.CHPH(FDA4H)
意味 MSXIO CHPUT(1文字出力)の始め
使用目的 他のコンソール出力装置をつなぐため

H.DSPC(FDA9H)
意味　MSXIO DSPCSR(カーソル表示)の始め
使用目的　他のコンソール出力装置をつなぐため

H.ERAC(FDAEH)
意味　MSXIO ERACSR(カーソル消去)の始め
使用目的　他のコンソール出力装置をつなぐため

H.DSPF(FDB3H)
意味　MSXIO DSPFNK(ファンクションキー表示)の始め
使用目的　他のコンソール出力装置をつなぐため

H.ERAF(FDB8H)
意味　MSXIO ERAFNK(ファンクションキー消去)の始め
使用目的　他のコンソール出力装置をつなぐため

H.TOTE(FDBDH)
意味　MSXIO TOTEXT(画面をテキストモードにする)の始め
使用目的　他のコンソール出力装置をつなぐため

H.CHGE(FDC2H)
意味　MSXIO CHGET(1文字取り出し)の始め
使用目的　他のコンソール出力装置をつなぐため

H.INIP(FDC7H)
意味　MSXIO INIPAT(文字パターンの初期化)の始め
使用目的　他の文字セットを使うため

H.KEYC(FDCCH)
意味　MSXIO KEYCOD(キーコード変換)の始め
使用目的　他のキー配置を使うため

H.KYEA(FDD1H)
意味　MSXIO NMIルーチン(Key Easy)の始め
使用目的　他のキー配置を使うため

H.NMI(FDD6H)
意味　MSXIO NMI(ノンマスカブルインタラプト)の始め
使用目的　NMI処理をするため

H.PINL(FDDBH)
意味　MSXINL PINLIN(1行入力)の始め
使用目的　他のコンソール入力装置や他の入力方式を使うため

H.QINL(FDE0H)
意味　MSXINL QINLIN("?"を表示して1行入力)の始め
使用目的　他のコンソール入力装置や他の入力方式を使うため

H.INLI(FDE5H)
意味　MSXINL INLIN(1行入力)の始め
使用目的　他のコンソール入力装置や他の入力方式を使うため

H.ONGO(FDEAH)
意味　MSXSTS INGOTP(ON GOTO)の始め
使用目的　他の割り込み処理装置を使うため

H.DSKO(FDEFH)
意味　MSXSTS DSKO$(ディスク出力)の始め
使用目的　ディスク装置を接続するため

H.SETS(FDF4H)
意味　MSXSTS SETS(セット　アトリビュート)の始め
使用目的　ディスク装置を接続するため

H.NAME(FDF9H)
意味　MSXSTS NAME(リネーム)の始め
使用目的　ディスク装置を接続するため

H.KILL(FDFEH)
意味　MSXSTS KILL(ファイルの削除)の始め
使用目的　ディスク装置を接続するため

H.IPL(FE03H)
意味　MSXSTS IPL(初期プログラムのロード)の始め
使用目的　ディスク装置を接続するため

H.COPY(FE08H)
意味　MSXSTS COPY(ファイルのコピー)の始め
使用目的　ディスク装置を接続するため

H.CMD(FE0DH)
意味　MSXSTS CMD(拡張コマンド)の始め
使用目的　ディスク装置を接続するため

H.DSKF(FE12H)
意味　MSXSTS DSKF(ディスクの空き)の始め
使用目的　ディスク装置を接続するため

H.DSKI(FE17H)
意味　MSXSTS DSKI(ディスク入力)の始め
使用目的　ディスク装置を接続するため

H.ATTR(FE1CH)
意味　MSXSTS ATTR$(アトリビュート)の始め
使用目的　ディスク装置を接続するため

H.LSET(FE21H)
意味 MSXSTS LSET(左詰め代入)の始め
使用目的 ディスク装置を接続するため

H.RSET(FE26H)
意味 MSXSTS RSET(左詰め代入)の始め
使用目的 ディスク装置を接続するため

H.FIEL(DE2BH)
意味 MSXSTS FIELD(フィールド)の始め
使用目的 ディスク装置を接続するため

H.MKI$(FE30H)
意味 MSXSTS MKI$(整数作成)の始め
使用目的 ディスク装置を接続するため

H.MKS$(FE35H)
意味 MSXSTS MKS$(単精度実数作成)の始め
使用目的 ディスク装置を接続するため

H.MKD$(FE3AH)
意味 MSXSTS MKD$(倍精度実数作成)の始め
使用目的 ディスク装置を接続するため

H.CVI(FE3FH)
意味 MSXSTS CVI(整数変換)の始め
使用目的 ディスク装置を接続するため

H.CVS(FE44H)
意味 MSXSTS CVS(単精度実数変換)の始め
使用目的 ディスク装置を接続するため

H.CVD(FE49H)
意味 MSXSTS CVD(倍精度実数変換)の始め
使用目的 ディスク装置を接続するため

H.GETP(FE4EH)
意味 SPDSK GETPTR(ファイルポインタ取り出し)
使用目的 ディスク装置を接続するため

H.SETF(FE53H)
意味 SPCDSK SETFIL(ファイルポインタ設定)
使用目的 ディスク装置を接続するため

H.NOFO(FE58H)
意味 SPDSK NOFOR(OPEN 文に FOR がない)
使用目的 ディスク装置を接続するため

H.NULO(FE5DH)
意味 SPCDSK NULOPN(空きファイルをオープン)
使用目的 ディスク装置を接続するため

H.NTFL(FE62H)
意味 SPCDSK NTFLO(ファイル番号が0でない)
使用目的 ディスク装置を接続するため

H.MERG(FE67H)
意味 SPCDSK MERGE(プログラムファイルのマージ)
使用目的 ディスク装置を接続するため

H.SAVE(FE6CH)
意味 SPCDSK SAVE(セーブ)
使用目的 ディスク装置を接続するため

H.BINS(FE71H)
意味 SPCDSK BINSAV(機械語セーブ)
使用目的 ディスク装置を接続するため

H.BINL(FE76H)
意味 SPCDSK BINLOD(機械語ロード)
使用目的 ディスク装置を接続するため

H.FILE(FD7BH)
意味 SPCDSK FILES(ファイル名の表示)
使用目的 ディスク装置を接続するため

H.DGET(FE80H)
意味 SPCDSK DGET(ディスク GET)
使用目的 ディスク装置を接続するため

H.FILO(FE85H)
意味 SPCDSK FILOU1(ファイル出力)
使用目的 ディスク装置を接続するため

H.INDS(FE8AH)
意味 SPCDSK INDSKC(ディスクの属性を入力)
使用目的 ディスク装置を接続するため

H.RSLF(FE8FH)
意味 SPCDSK 前のドライブを再び選択する
使用目的 ディスク装置を接続するため

H.SAVD(FE94H)
意味 SPCDSK 現在選択しているドライブを保存する
使用目的 ディスク装置を接続するため

H.LOC(FE99H)
意味 SPCDSK LOC 関数(場所を示す)
使用目的 ディスク装置を接続するため

H.LOF(FE9EH)
意味 SPCDSK LOF 関数(ファイルの長さ)
使用目的 ディスク装置を接続するため

H.EOF(FEA3H)
意味 SPCDSK EOF 関数(ファイルの終わり)
使用目的 ディスク装置を接続するため

H.FPOS(FEA8H)
意味　SPCDSK FPOS関数(ファイルの場所)
使用目的　ディスク装置を接続するため
H.BAKU(FEADH)
意味　SPCDSK BAKUPT(バックアップ)
使用目的　ディスク装置を接続するため
H.PARD(FEB2H)
意味　SPCDEV PARDEV(装置名の取り出し)
使用目的　論理装置名を拡張するため
H.NODE(FEB7H)
意味　SPCDEV NODEVN(装置名なし)
使用目的　省略装置名を他の装置に設定する
H.POSD(FEBCH)
意味　SPCDEV POSDSK
使用目的　ディスク装置を接続するため
H.DEVN(FEC1H)
意味　SPCDEV DEVNAM(装置名の処理)
使用目的　論理装置名を拡張するため
H.GEND(FEC6H)
意味　SPCDEV GENDSP(装置割り当て)
使用目的　論理装置名を拡張するため
H.RUNC(FECBH)
意味　BIMISC RUNC(RUNのためのクリア)
H.CLEA(FED0H)
意味　BIMISC CLEARC(CLEAR文のためのクリア)
H.LOPD(FED5H)
意味　BIMISC LOPDFT (繰り返しと省略値の設定)
使用目的　変数に他の省略値を使うため
H.STKE(FEDAH)
意味　BIMISC STKERR(スタックエラー)
H.ISFL(FEDFH)
意味　BIMISC ISFLIO(ファイルの入出力かどうか)
H.OUTD(FEE4H)
意味　BIO OUTDO(OUTを実行)
H.CRDO(FEE9H)
意味　BIO CRDO(CRLFを実行)
H.DSKC(FEEEH)
意味　BIO DSKCHI(ディスクの属性を入力)
H.DOGR(FEF3H)
意味　GENGRP DOGRPH(グラフィック処理を実行)
H.PRGE(FEF8H)
意味　BINTRP PRGEND(プログラム終了)
H.ERRP(FEFDH)
意味　BINTRP ERRPRT(エラー表示)
H.ERRF(FF02H)
意味　BINTRP
H.READ(FF07H)
意味　BINTRP READY
H.MAIN(FF0CH)
意味　BINTRP MAIN
H.DIRD(FF11H)
意味　BINTRP DIRDO(ダイレクトステートメント実行)
H.FINI(FF16H)
意味　BINTRP
H.FINE(FF1BH)
意味　BINTRP
H.CRUN(FF20H)
意味　BINTRP
H.CRUS(FF25H)
意味　BINTRP
H.ISRE(FF2AH)
意味　BINTRP
H.NTFN(FF2FH)
意味　BINTRP
H.NOTR(FF34H)
意味　BINTRP
H.SNGF(FF39H)
意味　BINTRP
H.NEWS(FF3EH)
意味　BINTRP
H.GONE(FF43H)
意味　BINTRP
H.CHRG(FF48H)
意味　BINTRP
H.RETU(FF4DH)
意味　BINTRP
H.PRTF(FF52H)
意味　BINTRP
H.COMP(FF57H)
意味　BINTRP
H.FINP(FF5CH)
意味　BINTRP
H.TRMN(FF61H)
意味　BINTRP
H.FRME(FF66H)
意味　BINTRP
H.NTPL(FF6BH)
意味　BINTRP

H.EVAL(FF70H)
　意味　BINTRP
H.OKNO(FF75H)
　意味　BINTRP
H.FING(FF7AH)
　意味　BINTRP
H.ISMI(FF7FH)
　意味　BINTRP ISMID$(MID$かどうか)
H.WIDT(FF84H)
　意味　BINTRP WIDTHS(WIDTH)
H.LIST(FF89H)
　意味　BINTRP LIST
H.BUFL(FF8EH)
　意味　BINTRP BUFLIN(バッファライン)
H.FRQI(FF93H)
　意味　BINTRP FRQINT
H.SCNE(FF98H)
　意味　BINTRP
H.FRET(FF9DH)
　意味　BISTRS FRETMP
H.PTRG(FFA2H)
　意味　BIPTRG PTRGET(ポインタ取り出し)
　使用目的　省略値以外の変数を使用するため
H.PHYD(FFA7H)
　意味　MSXIO PHYDIO(物理ディスク入出力)
　使用目的　ディスク装置を接続するため
H.FORM(FFACH)
　意味　MSXIO FORMAT(ディスクをフォーマットする)
　使用目的　ディスク装置を接続するため
H.ERRO(FFB1H)
　意味　BINTRP ERROR
　使用目的　アプリケーション・プログラムのエラー処理
H.LPTO(FFB6H)
　意味　MSXIO LPTOUT　(プリンタ出力)
　使用目的　省略値以外のプリンタを使うため
H.LPTS(FFBBH)
　意味　MSXIO LPTSTT(プリンタの状態)
　使用目的　省略値以外のプリンタを使うため
H.SCRE(FFC0H)
　意味　MSXSTS SCREEN 文の入口
　使用目的　SCREEN 文を拡張するため
H.PLAY(FFC5H)
　意味　MSXSTS PLAY 文の入口
　使用目的　PLAY 文を拡張するため

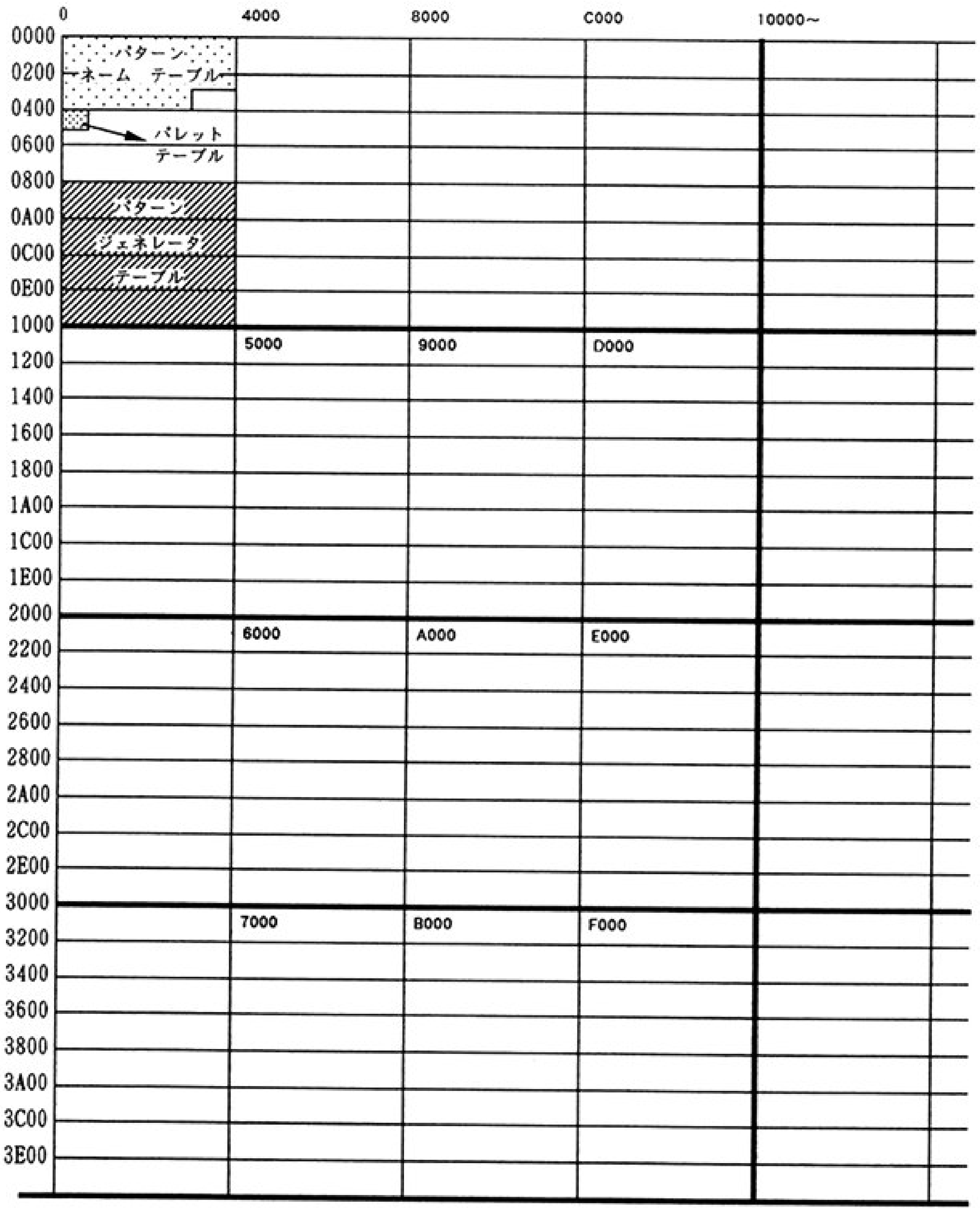
●拡張 BIOS 用

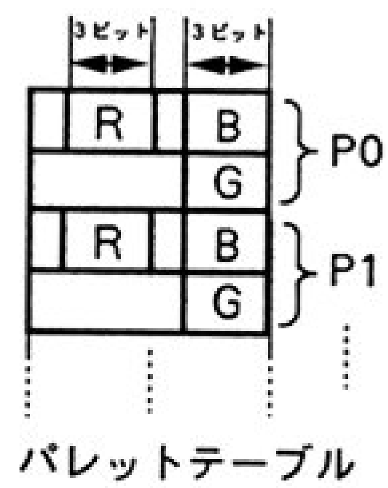
FCALL(FFCAH)
　内容　拡張BIOSが使用するフック
DISINT(FFCFH)
　内容　DOS が使用する
ENAINT(FFD4H)
　内容　DOS が使用する

A.5 VRAMマップ

SCREEN 0, (WIDTH40)

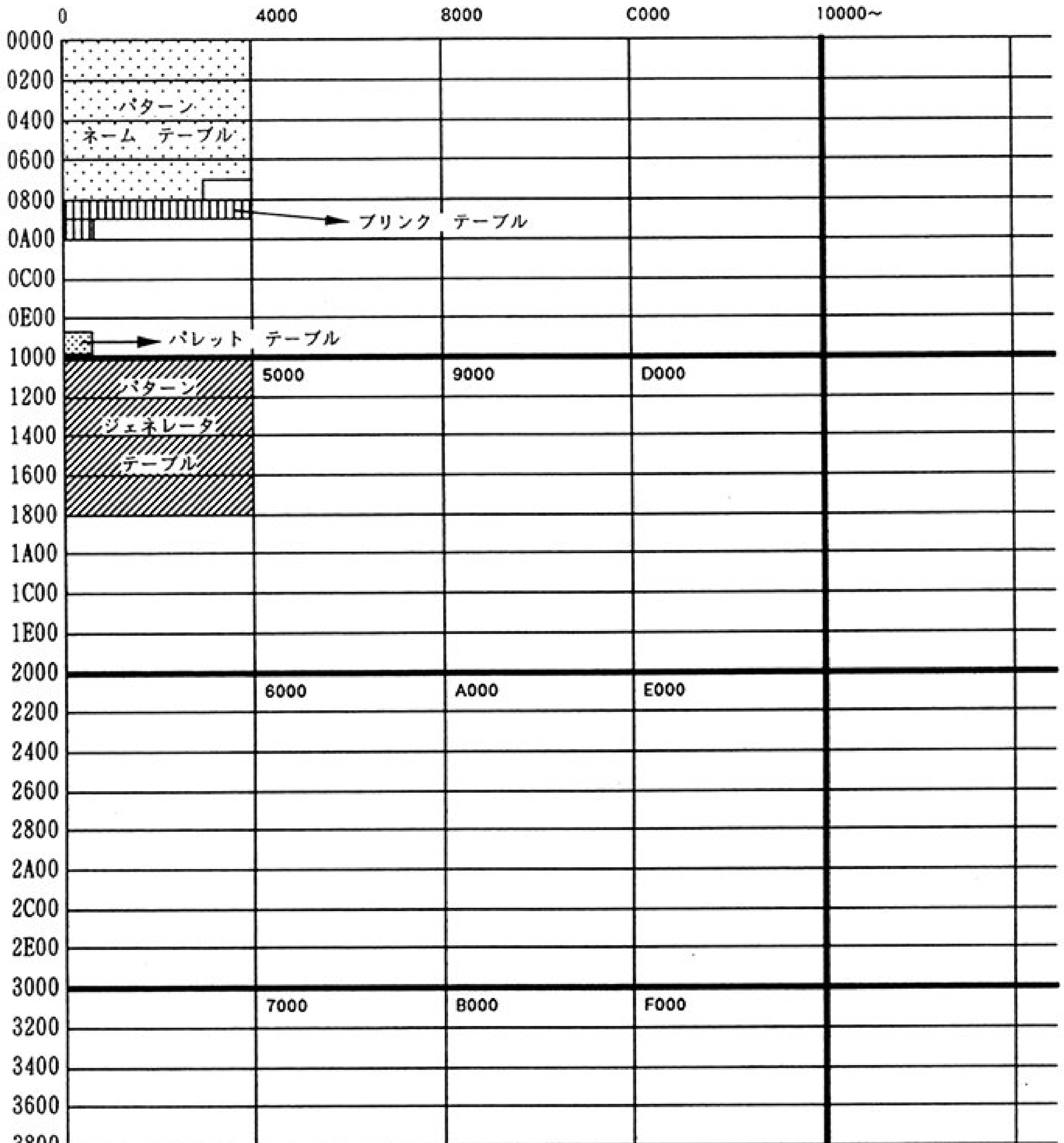
TEXT 1





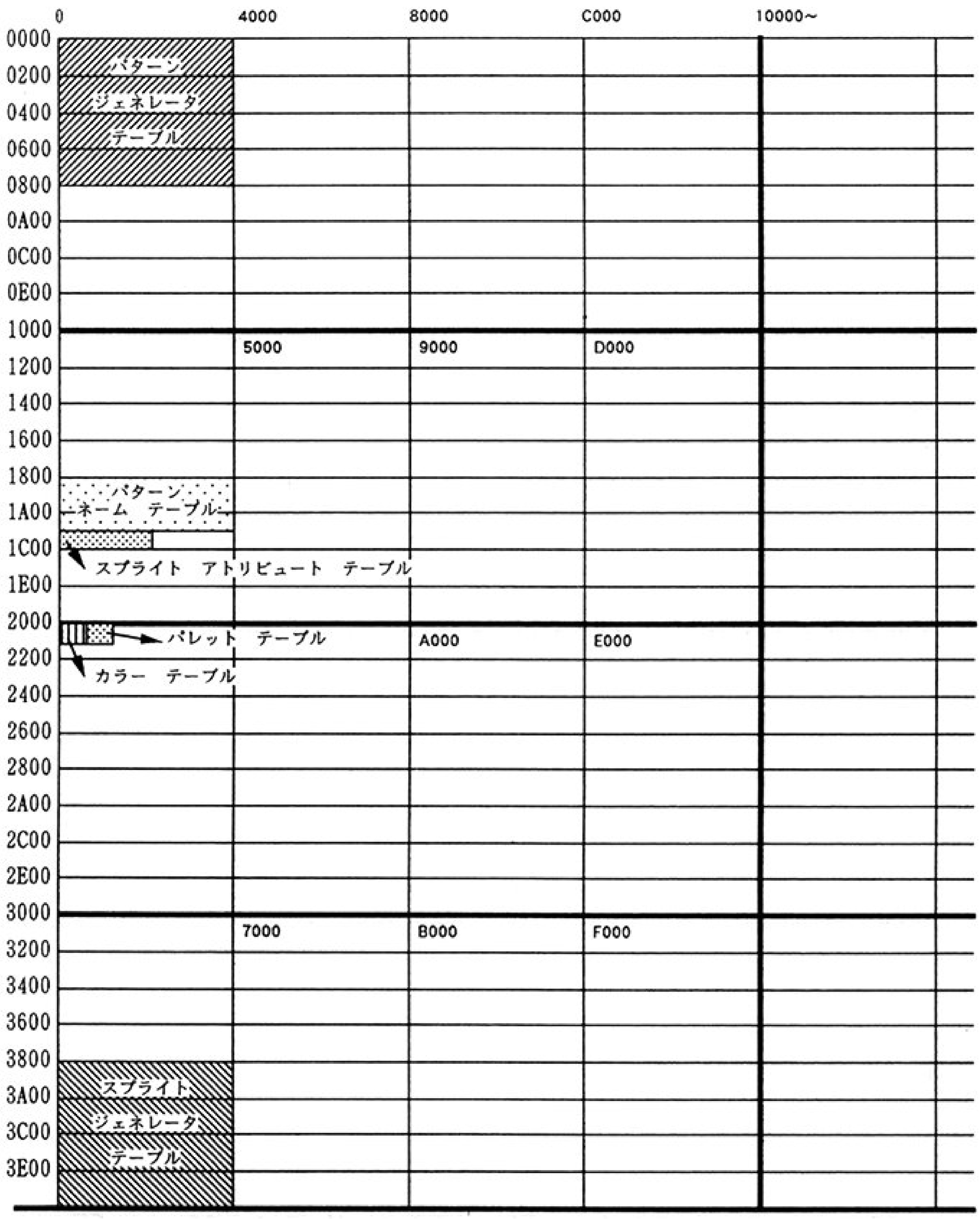
SCREEN 0, (WIDTH80)

TEXT 2



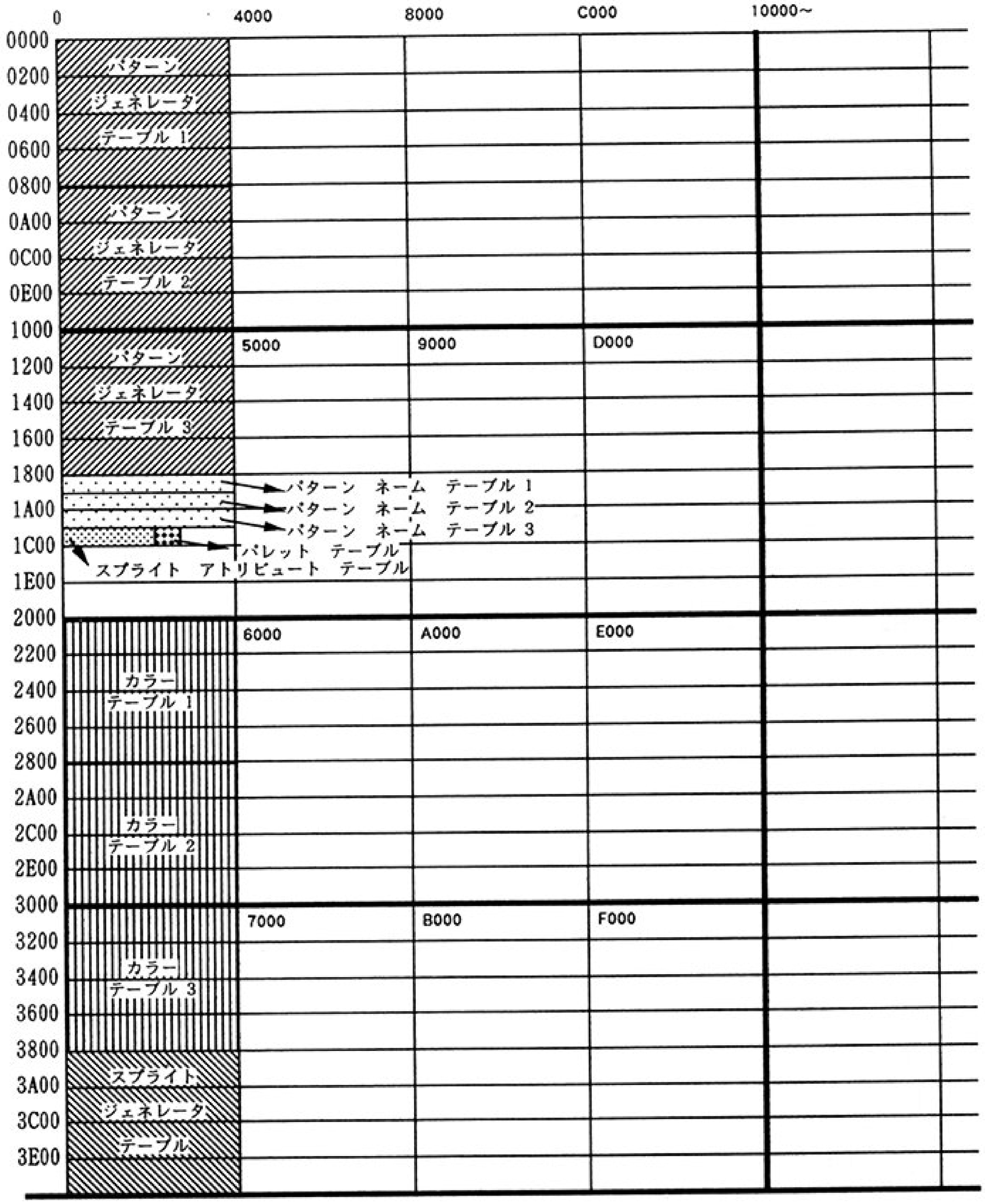
S C R E E N 1

GRAPHIC 1



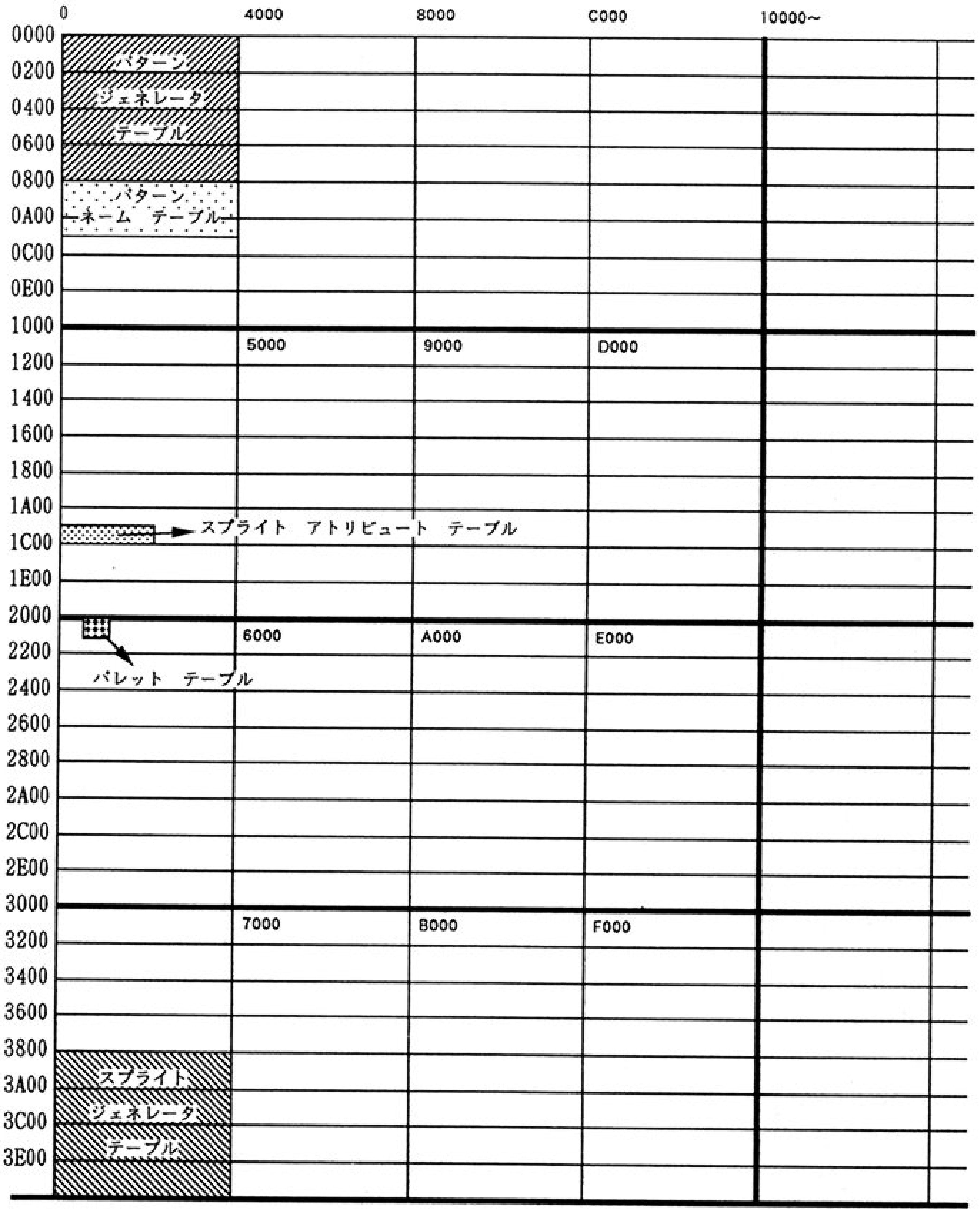
S C R E E N　2

GRAPHIC 2



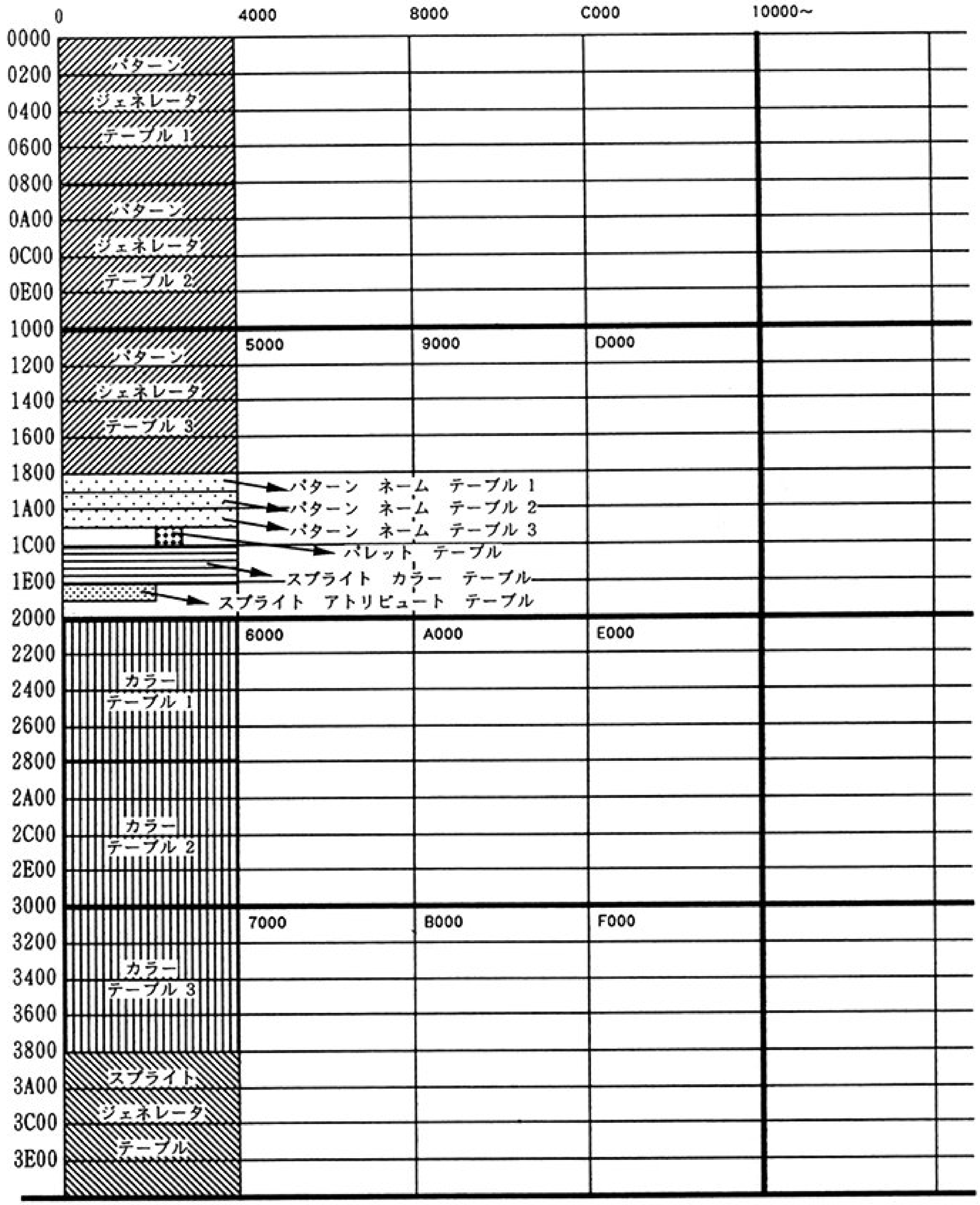
SCREEN 3

MULTI COLOR



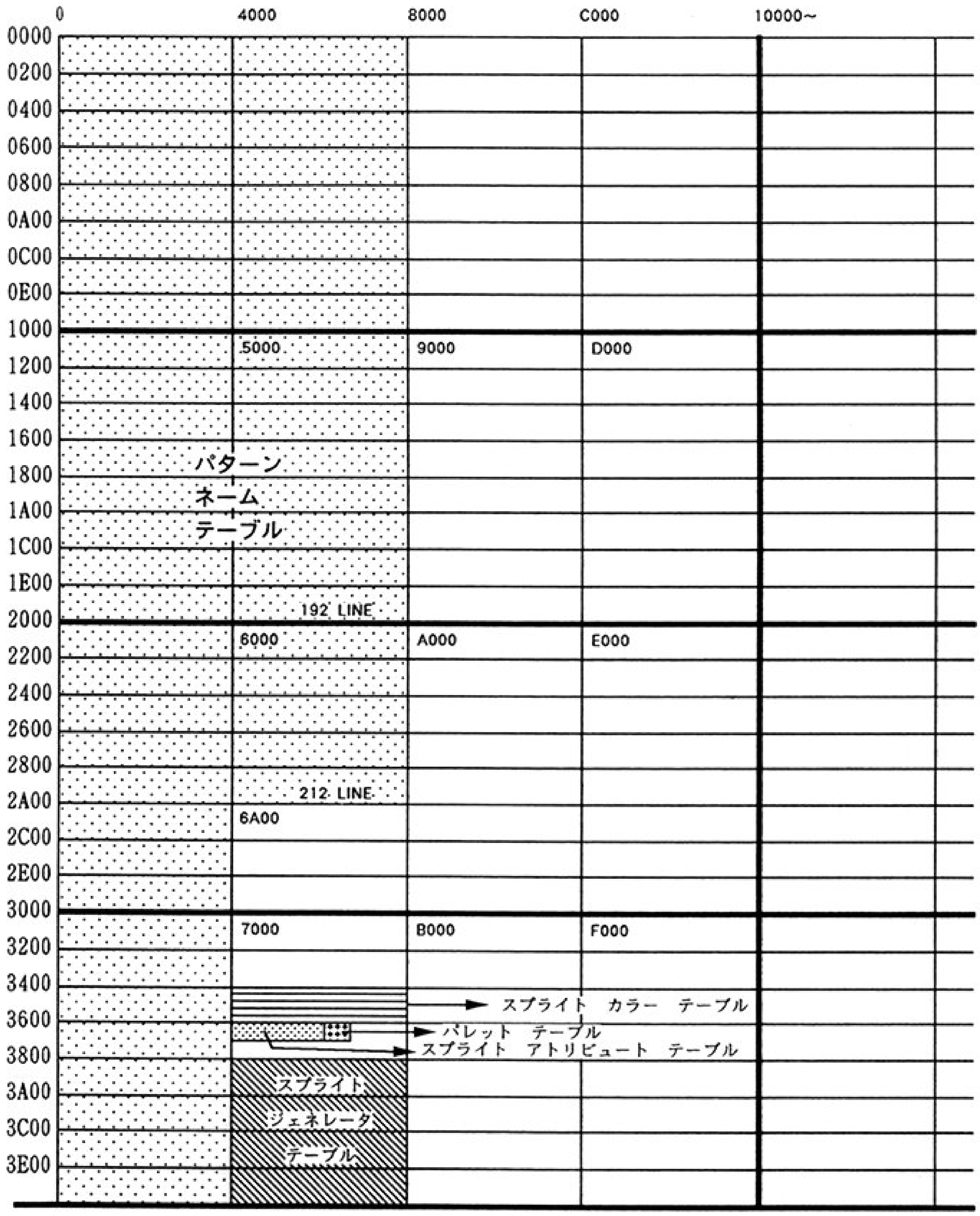
SCREEN 4

GRAPHIC 3



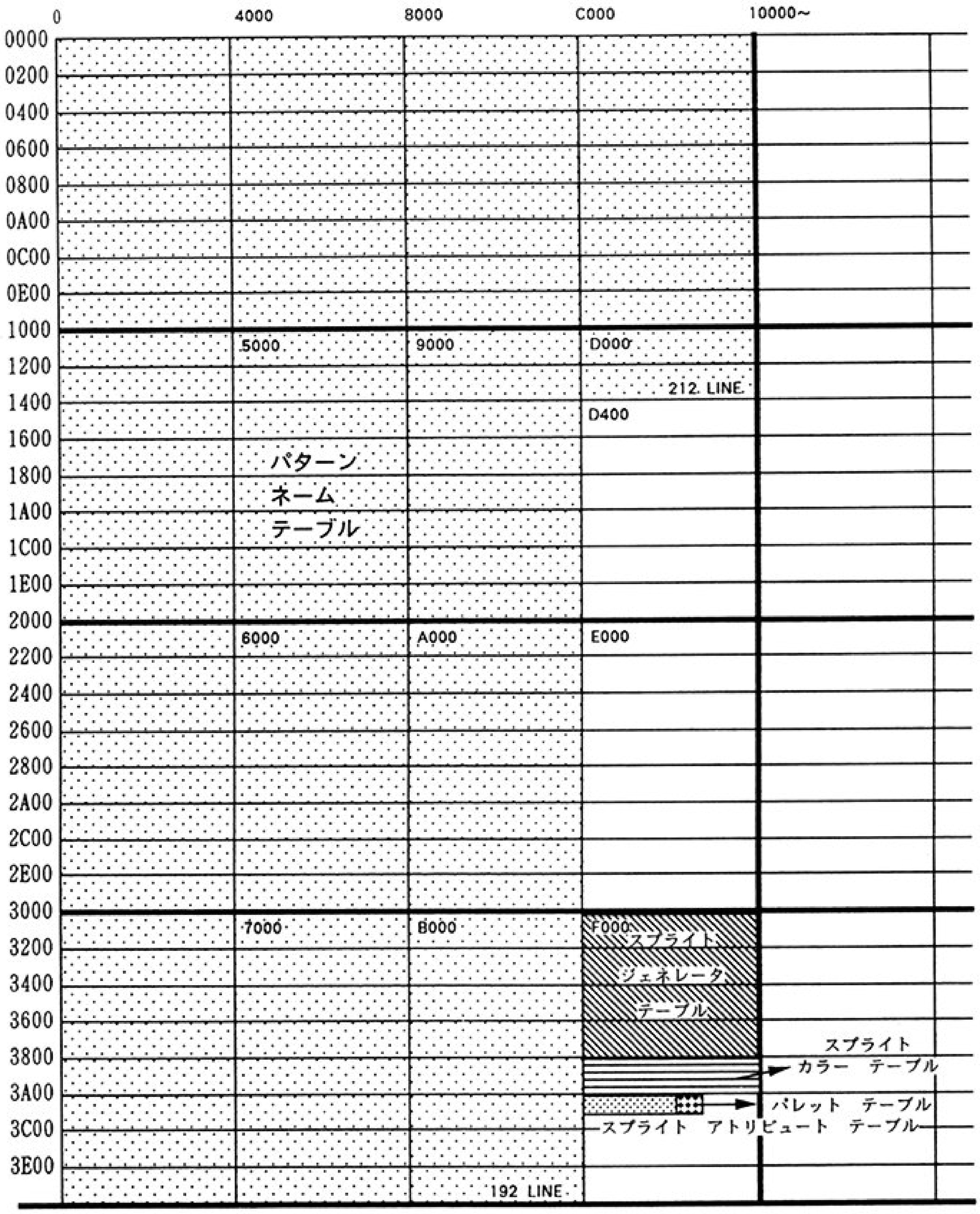
SCREEN 5, 6

GRAPHIC 4.5



SCREEN 7,8

GRAPHIC 6.7



A.6 I／Oマップ

00H～3FH	ユーザー定義
40H～7FH	リザーブ
80H～87H	RS-232C用
80H	8251 データ
81H	8251 ステータス／コマンド
82H	ステータスリード／インターラプトマスク
83H	未使用
84H	8253
85H	8253
86H	8253
87H	8253
88H～8BH	MSX1アダプタのためのVDP(V9938)用I／Oポート ここはMSX1にV9938をのせたときのI／Oとなります.VDPを直接I／Oアクセスする場合は必ずメインROMの6番地と7番地を調べ，ポートアドレスを確認して下さい.
8CH～8DH	モデム用
8EH～8FH	リザーブ
90H～91H	プリンタポート
90H	ビット0：ストローブ出力(ライト) ビット1：ステータス入力(リード)
91H	プリントデータ
92H～97H	リザーブ
98H～9BH	VDP(V9938)MSX2用
98H	VRAMアクセス
99H	コマンドレジスタアクセス
9AH	パレットレジスタアクセス(ライトのみ)
9BH	レジスタ間接指定(ライトのみ)
9CH～9FH	リザーブ
A0H～A3H	サウンドジェネレータ(AY-3-8910)
A0H	アドレスラッチ
A1H	データライト
A2H	データリード
A4H～A7H	リザーブ
A8H～ABH	パラレルポート(8255)
A8H	ポートA
A9H	ポートB
AAH	ポートC
ABH	モードセット
ACH～AFH	MSXエンジン(1チップMSX I／O)
B0H～B3H	拡張メモリ(ソニー仕様)(8255)
B0H	ポートA，アドレス(A0-A7)
B1H	ポートB，アドレス(A8-A10，A13-A15)，コントロール，R／W
B2H	ポートC，アドレス(A11-A12)，データ(D0-D7)
B3H	モードセット
B4H～B5H	CLOCK-IC(RP-5C01)
B4H	アドレスラッチ
B5H	データ

B6H～B7H	リザーブ
B8H～BBH B8H B9H BAH BBH	ライトペンコントロール（サンヨー仕様） リード／ライト リード／ライト リード／ライト ライトのみ
BCH～BFH BCH BDH BEH BFH	VHDコントロール（JVC）（8255） ポートA ポートB ポートC モードセット
C0H～C1H	MSXオーディオ
C2H～C7H	リザーブ
C8H～CFH	MSXインターフェイス
D0H～D7H	フロッピーディスクコントローラ（FDC） フロッピーディスクコントローラは外部からの信号によって遮断される機能を持つような仕様になっており，FDCがアクセスされるときに限って使用可能となります．この機構により，システムが複数の異なるFDCインターフェイスを扱うことが可能になります．
D8H～D9H D8H D9H	漢字ROM（東芝仕様） b5-b0 下位アドレス（ライトのみ） b5-b0 上位アドレス（ライト） b7-b0 データ（リード）
DAH～DBH	漢字のための将来の拡張用
DCH～F4H	リザーブ
F5H b0 b1 b2 b3 b4 b5 b6 b7	システムコントロール（ライトのみ） ビットを1にすることによりI／Oデバイスが使用可能になります 漢字ROM 漢字のためリザーブ MSXオーディオ スーパーインポーズ MSXインターフェイス RS-232C ライトペン CLOCK-IC（MSX2のみ実装） 内蔵あるいはカートリッジにより接続されたI／Oデバイスの衝突をさけるためのビット．これによって内蔵のデバイスを無効にすることができます．BIOSの初期化時に，外部デバイスが無ければ内部デバイスを有効にします．ここは，アプリケーションプログラムが読み書きしてはいけません．
F6H	カラーバスI／O
F7H b0 b1 b2 b3 b4 b5 b6 b7	A／Vコントロール オーディオR　L：ミキシング ON （ライト） オーディオL　L：ミキシング OFF （ライト） ビデオ入力の選択　L：21p RGB （ライト） ビデオ入力の検出　L：入力なし （リード） AVコントロール　L：TV （ライト） Ymコントロール　L：TV （ライト） VDPレジスター9のビット4の反転 （ライト） VDPレジスター9のビット5の反転 （ライト）
F8H～FBH	リザーブ
FCH～FFH	メモリーマッパー

A.7 カートリッジハードウェア仕様

第5部7章ではカートリッジROM内に収めるソフトウェアの作成法を説明しましたが，実際にカートリッジを製作する場合には，ここに示したハードウェア仕様を参考にしてください．

```
;============================================================
;  List A.1    MSX ROM Cartridge Header to start by H.STKE
;              (this program must link List 5.15)
;============================================================
;
;       Switch
;
SNSDEL  EQU     1               ;Sense DEL key not to start
;
;       BIOS
;
SNSMAT  EQU     141H            ;Sense key in real time
;
;       System work area
;
H.STKE  EQU     0FEDAH          ;Hook for auto start
;
;       External symbol
;
        EXTRN   START           ;Address to start
        EXTRN   GTSL1@          ;Get my slot address
;
;       Header
;
        ASEG
        ORG     4000H
        DB      41H, 42H
        DW      INIT
        DW      0               ;No STATEMENT
        DW      0               ;No DEVICE
        DW      0               ;No TEXT
        DW      0
        DW      0
        DW      0
;
;       INIT : set H.STKE and return
;
INIT:
        IF      SNSDEL
        LD      A,8             ;Read 8th row
        CALL    SNSMAT          ;Read key
        BIT     3,A             ;DEL key
        RET     Z               ;Return if DEL pressed
        ENDIF                   ;IF SNSDEL
```

```
;
        LD      HL, H.STKE      ;Hook for auto start
        LD      A, 0F7H         ;"RST 30H" instruction
        DI                      ;For fale safe
        LD      (HL), A
        CALL    GTSL1@          ;[A] = my slot address
        INC     HL
        LD      (HL), A
        LD      DE, START       ;Set start address
        INC     HL
        LD      (HL), E
        INC     HL
        LD      (HL), D
        EI
        RET

        END
```

```
;=================================================================
; List A.2   MSX ROM Cartridge Header to get work area from HIMEM
;=================================================================
        .SFCOND
        .XALL
        .RADIX  10
;
;       Switch
;
SNSDEL  EQU     1               ;Sense DEL key not to start
;
;       Constant
;
MYWORK  EQU     1024            ;Length of my work area to request
;
;       BIOS
;
RSLREG  EQU     138H            ;Get primary slot register
SNSMAT  EQU     141H            ;Sense key in real time
CALBAS  EQU     159H            ;Call BASIC interpreter
;
;       System work area
;
MEMSIZ  EQU     0F672H
VARTAB  EQU     0F6C2H
STKTOP  EQU     0F674H
TEMP    EQU     0F6A7H          ;To save text pointer
MAXFIL  EQU     0F85FH
FILTAB  EQU     0F860H
NULBUF  EQU     0F862H
BOTTOM  EQU     0FC48H
HIMEM   EQU     0FC4AH
EXPTBL  EQU     0FCC1H
SLTWRK  EQU     0FD09H
H.CLEA  EQU     0FED0H          ;Hook to get work area
;
;       MACROS
;
COMPAR  MACRO
        RST     20H             ;Compares HL with DE
        ENDM
```

```
;
BIOS    MACRO   ADDRESS
        CALL    ADDRESS             ;Will be change for DOS version
        ENDM
;
;
;       Header
;
        CSEG
;        ASEG
;        ORG     4000H
        DB      41H, 42H
        DW      INIT
        DW      0                   ;No STATEMENT
        DW      0                   ;No DEVICE
        DW      0                   ;No TEXT
        DW      0
        DW      0
        DW      0
;
;       INIT : set H.CLEA and return
;
INIT:
        IF      SNSDEL
        LD      A,8                 ;Read 8th row
        CALL    SNSMAT              ;Read key
        BIT     3,A                 ;DEL key
        RET     Z                   ;Return if DEL pressed
        ENDIF                       ;IF SNSDEL
;
        LD      HL, H.CLEA          ;Hook to get work area
        LD      A, 0F7H             ;"RST 30H" instruction
        DI                          ;For fale safe
        LD      (HL), A
        CALL    GTSLI@              ;[A] = my slot address
        INC     HL
        LD      (HL), A
        LD      DE, START           ;Set start address
        INC     HL
        LD      (HL), E
        INC     HL
        LD      (HL), D
        EI
        RET
;
;       Called by H.CLEA
;       Set work area
;
START:
        LD      (TEMP),HL           ;Save [HL]
        DI                          ;For fail safe
        LD      B,5
        LD      HL,H.CLEA
        LD      A,0C9H              ;RET instruction
CLHOOK:
        LD      (HL),A              ;Clear H.CLEA
        INC     HL
        DJNZ    CLHOOK
        EI
;
        LD      HL,MYWORK           ;How many memory to request
        CALL    WORKH@              ;Change HIMEM
;
;       [HL] = (HIMEM) = start address of my work area
;       [HL] = 0 if out of memory
```

```
;
        CALL    DEFILE          ;Move stack and FCB
;
;       Note that DEFILE changes stack pointer
;
        LD      IX,7D20H        ;Return address
        JP      CALBAS          ;Go to BASIC.
;
;       Note that H.CLEA will be called again.
;       Make sure that H.CLEA is cleared by 'RET'.
;
;-----------------------------------------------------------
;       BOTTOM became greater than 0DFFFH, there is
;       no RAM left to be allocated.
;
NOROOM:
        LD      HL,0
        CALL    WSLW1@          ;Clear slot work
        JR      WORKBEND        ;Return 0 in [HL]
        ENDIF                   ;IF USEWORKB OR USEWORKH
        IF      USEWORKH
;
;       WORKH   allocate work area from HIMEM
;               (my slot work) <- (new HIMEM)
;       Entry   HL      required memory size
;       Return  HL      start address of my work erea = new HIMEM
;                       0 if can not allocate
;       Modify  None
;       Note    This routine shoud be called by H.STKE not by INIT.
;
LIMHIM  EQU     0C400H          ;Lower limit of HIMEM for fale safe
;
        PUBLIC  WORKH@
WORKH@:
        PUSH    DE
        PUSH    BC
        PUSH    AF
;
        EX      DE,HL           ;[DE] = Size
        LD      HL,(HIMEM)      ;Get current RAM top
        XOR     A               ;Clear carry
        SBC     HL, DE          ;[HL] = new HIMEM
        JR      C, NOROOM       ;HIMEM < 0
        EX      DE, HL          ;[DE] = new HIMEM
        LD      HL, LIMHIM      ;Lower limit of HIMEM
        XOR     A               ;Clear carry
        SBC     HL, DE
        JR      NC, NOROOM      ;HIMEM < LIMHIM
        EX      DE, HL          ;[HL] = new HIMEM
        CALL    WSLW1@          ;Save HIMEM to slot work
        LD      (HIMEM),HL      ;Update HIMEM
        JR      WORKBEND
        ENDIF                   ;IF USEWORKH
;
;       DEFILE : re-build file structure
;       Original program is in INIT.MAC
;       Changed to set workerea at H.CLEA
;       MAXFIL = 1
;       String space = 200
;       by nao-i on 3. Feb., 1986
;
;       Entry   (HIMEM)         New HIMEM
;               (STKTOP)        To know string space if not STR200
;               (MEMSIZ)        To know string space if not STR200
;               (VARTAB)        To check free space if CHKOME
;               [A]             New MAXFIL if not FILMAX1
```

```
;
;        Return   (MAXFIL)
;                 (FILTAB)
;                 (MEMSIZ)
;                 (STKTOP)
;                 (NULBUF)
;
;        Modify   AF, BC, DE, HL, SP
;
;        Note     This routine changes stack pointer.
;
;        Constant
;
NUMLEV   EQU      60
FILLEN   EQU      265
FL.BUF   EQU      9
;
;        Switch
;
MAXFIL1  EQU      1                  ;MAXFILE = 1
STR200   EQU      1                  ;String space is 200 Byte
CHKOME   EQU      0                  ;Check "Out of memory"
;
         IF       CHKOME
         EXTRN    OMERR              ;Entry of "Out of memory"
         ENDIF
;
DEFILE:
         IFE      MAXFIL1
         PUSH     AF                 ;Save new MAXFIL
         ENDIF                       ;IFE MAXFIL1
         LD       HL,(HIMEM)
         IF       MAXFIL1
         LD       DE,65536-FILLEN*2-4
         ADD      HL,DE
         ELSE
         LD       DE,65536-FILLEN-2
DEFIL1:
         ADD      HL,DE              ;[HL] = (HIMEM) - FILLEN - 2
         DEC      A
         JP       P,DEFIL1
         ENDIF                       ;IF MAXFIL1
         EX       DE,HL              ;DE:=FILTAB
         IF       STR200             ;String space is fixed
         LD       HL,200
         ELSE
         LD       BC,(STKTOP)        ;Get current string space in BC
         LD       HL,(MEMSIZ)
         XOR      A                  ;Clear carry
         SBC      HL,BC              ;[HL] = (MEMSIZ) - (STKTOP)
         ENDIF                       ;IF STR200
         IFE      MAXFIL1
         POP      AF                 ;Restore new MAXFIL
         ENDIF                       ;IFE MAXFIL1
         PUSH     HL                 ;Save current string space
         IFE      MAXFIL1
         PUSH     AF                 ;Save new MAXFIL
         ENDIF                       ;IFE MAXFIL1
         IF       CHKOME
         LD       BC,2*NUMLEV+20
         ADD      HL,BC
         LD       B,H
         LD       C,L
         LD       HL,(VARTAB)        ;Get end of program
         ADD      HL,BC
```

```
        COMPAR                          ;Enough breathing room?
        JP      NC,OMERR                ;No
        ENDIF                           ;IF CHKOME
        IF      MAXFIL1
        LD      A, 1                    ;Make sure MAXFIL = 1
        ELSE
        POP     AF                      ;Restore new MAXFIL
        ENDIF                           ;IF MAXFIL1
        LD      (MAXFIL),A
        LD      L,E
        LD      H,D                     ;[HL] = [DE] = (HIMEM) - FILLEN - 2
        LD      (FILTAB),HL             ;Pointer to first FCB
        DEC     HL
        DEC     HL                      ;[HL] = (HIMEM) - FILLEN - 4
        LD      (MEMSIZ),HL
        POP     BC                      ;Restore current string space
        XOR     A                       ;Clear carry
        SBC     HL, BC
        LD      (STKTOP),HL             ;[HL] = (HIMEM) - FILLEN - 4 - <string size>
        DEC     HL
        DEC     HL
        POP     BC                      ;Get return address
        LD      SP,HL                   ;Set new temporary stack
        PUSH    BC
        IF      MAXFIL1
        LD      A, 1                    ;[A] = MAXFIL
        LD      HL, 4                   ;[HL] = MAXFIL * 2 + 2
        ELSE
        LD      A,(MAXFIL)
        LD      L,A
        INC     L
        LD      H,0
        ADD     HL,HL                   ;[HL] = (MAXFIL) * 2 + 2
        ENDIF                           ;IF MAXFIL1
        ADD     HL,DE                   ;Add FILTAB to get start of FCB's
        EX      DE,HL                   ;DE=Address of first FCB
        PUSH    DE                      ;Address of first FCB (#0)
        LD      BC,FILLEN
DSKFLL:
        LD      (HL),E
        INC     HL
        LD      (HL),D
        INC     HL
        EX      DE,HL
        LD      (HL),0                  ;Make it look closed
        ADD     HL,BC
        EX      DE,HL
        DEC     A
        JP      P,DSKFLL
        POP     HL                      ;Restore address of first FCB
        LD      BC,FL.BUF
        ADD     HL,BC                   ;Get address of buffer for file #0
        LD      (NULBUF),HL
        RET

        END
```

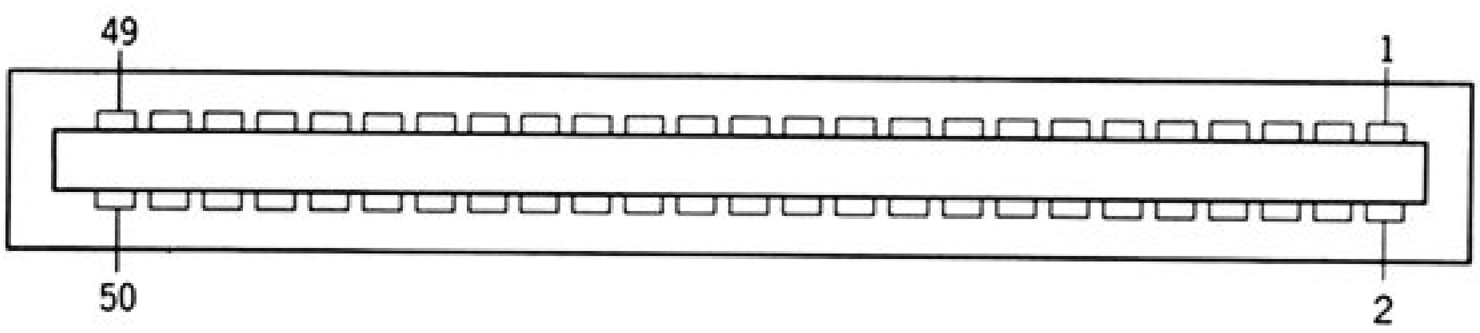
●カートリッジバス信号線表

ピンNo.	名　称	I／O※	ピンNo.	名　称	I／O※
1	$\overline{\text{CS1}}$	O	2	$\overline{\text{CS2}}$	O
3	$\overline{\text{CS12}}$	O	4	$\overline{\text{SLTSL}}$	O
5	予備　※1	−	6	$\overline{\text{RFSH}}$	O
7	$\overline{\text{WAIT}}$　※2	I	8	$\overline{\text{INT}}$　※2	I
9	$\overline{\text{M1}}$	O	10	$\overline{\text{BUSDIR}}$	I
11	$\overline{\text{IORQ}}$	O	12	$\overline{\text{MERQ}}$	O
13	$\overline{\text{WR}}$	O	14	$\overline{\text{RD}}$	O
15	$\overline{\text{RESET}}$	O	16	予備　※1	−
17	A9	O	18	A15	O
19	A11	O	20	A10	O
21	A7	O	22	A6	O
23	A12	O	24	A8	O
25	A14	O	26	A13	O
27	A1	O	28	A0	O
29	A3	O	30	A2	O
31	A5	O	32	A4	O
33	D1	I／O	34	D0	I／O
35	D3	I／O	36	D2	I／O
37	D5	I／O	38	D4	I／O
39	D7	I／O	40	D6	I／O
41	GND	−	42	CLOCK	O
43	GND	−	44	SW1	−
45	+5V	−	46	SW2	−
47	+5V	−	48	+12V	−
49	SUNDIN	I	50	−12V	−

注）本体を基準にした入出力の区別

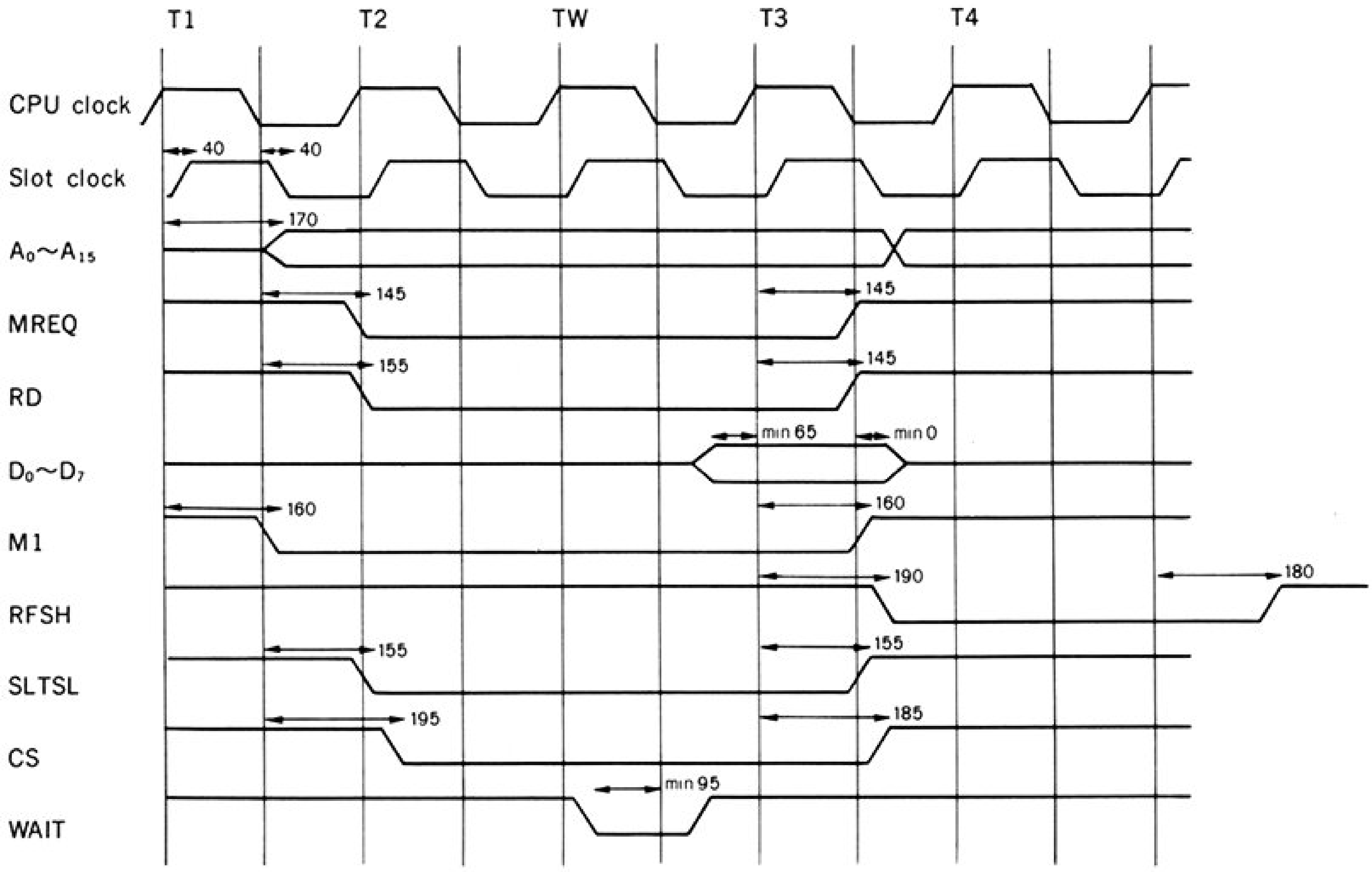
※1　予備は使用禁止端子

※2　OPEN COLLECTER出力

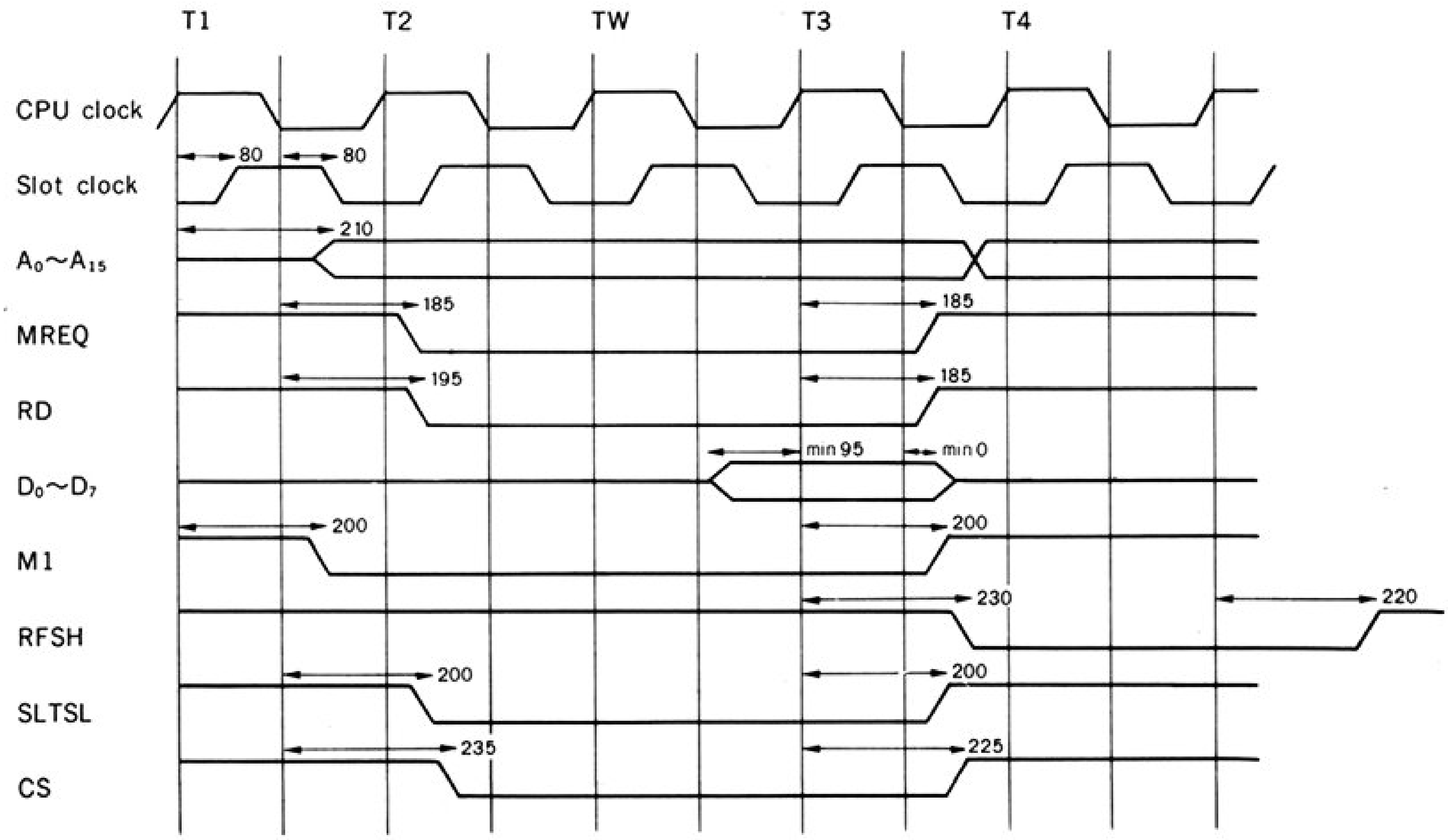


●スロットのタイミングチャート

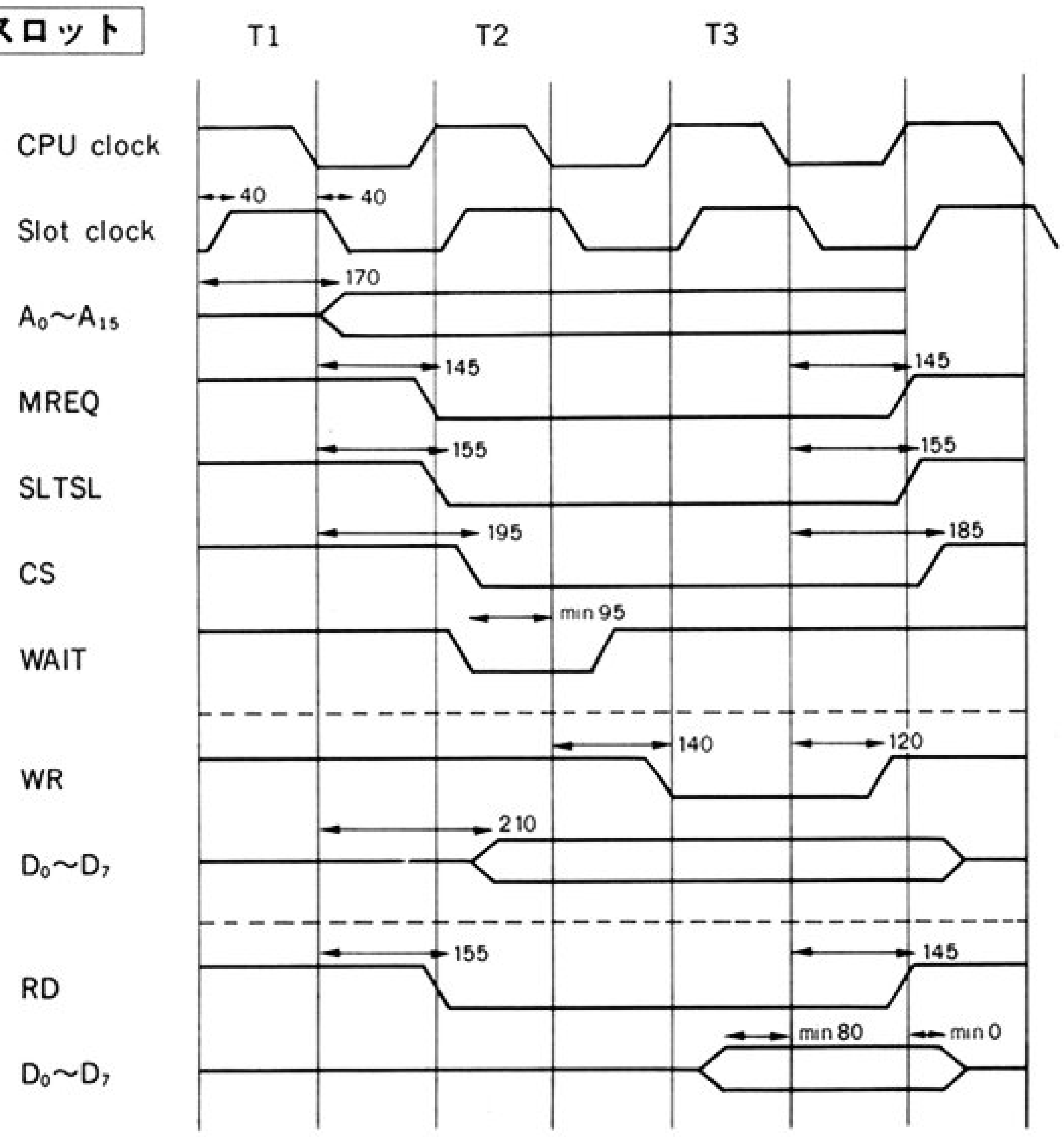
M1サイクル 基本スロット



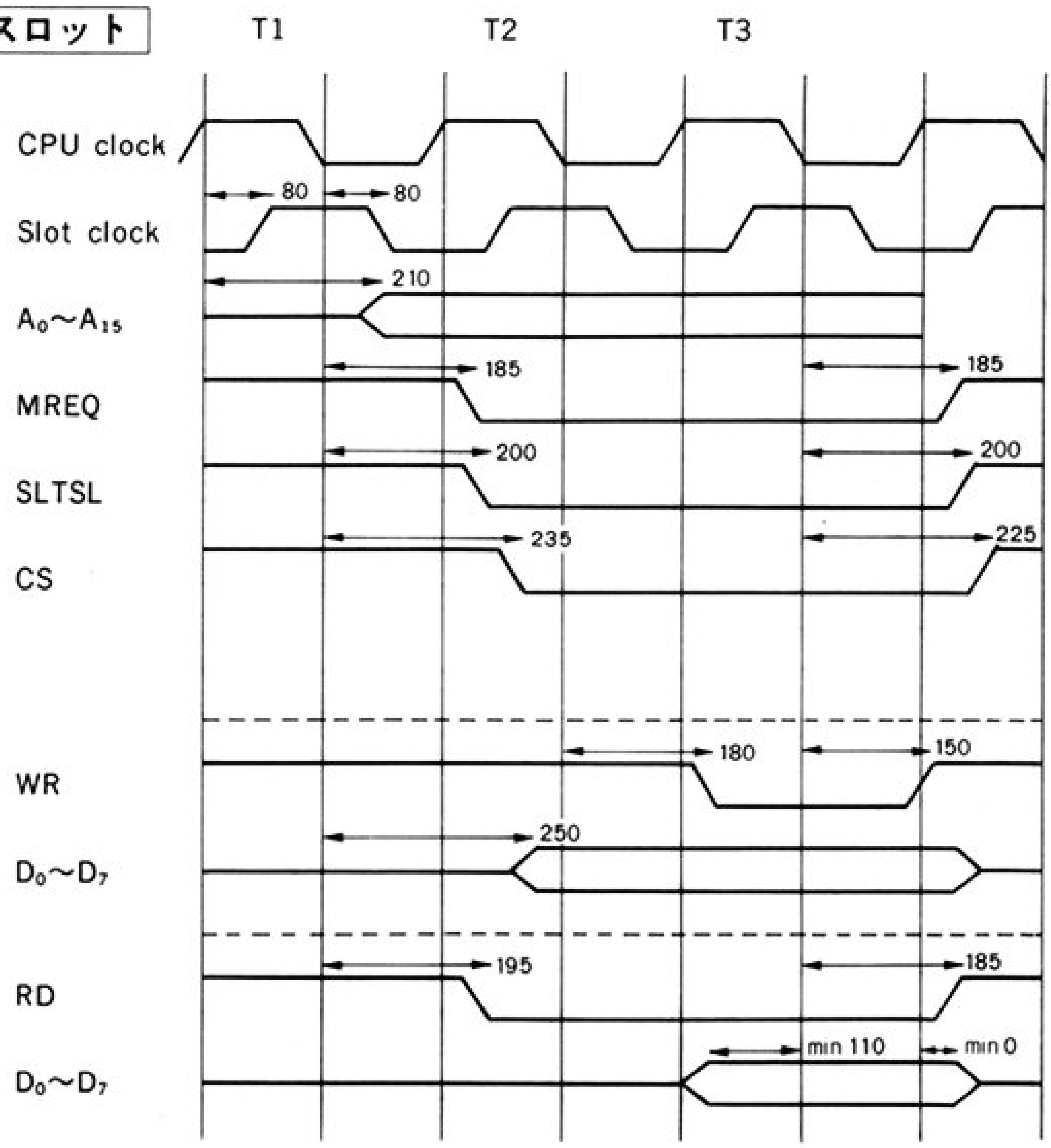
M1サイクル 拡張スロット



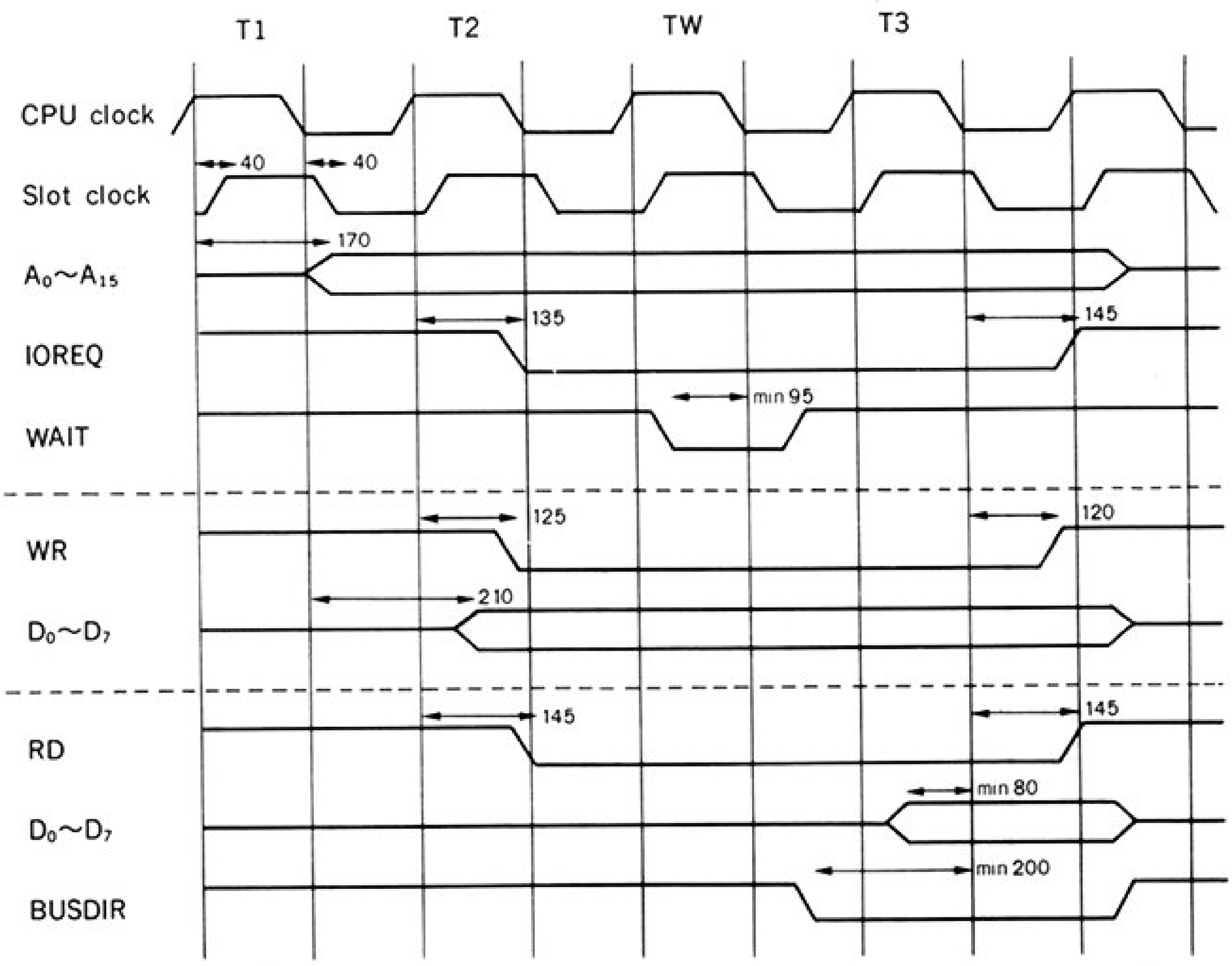
メモリサイクル 基本スロット



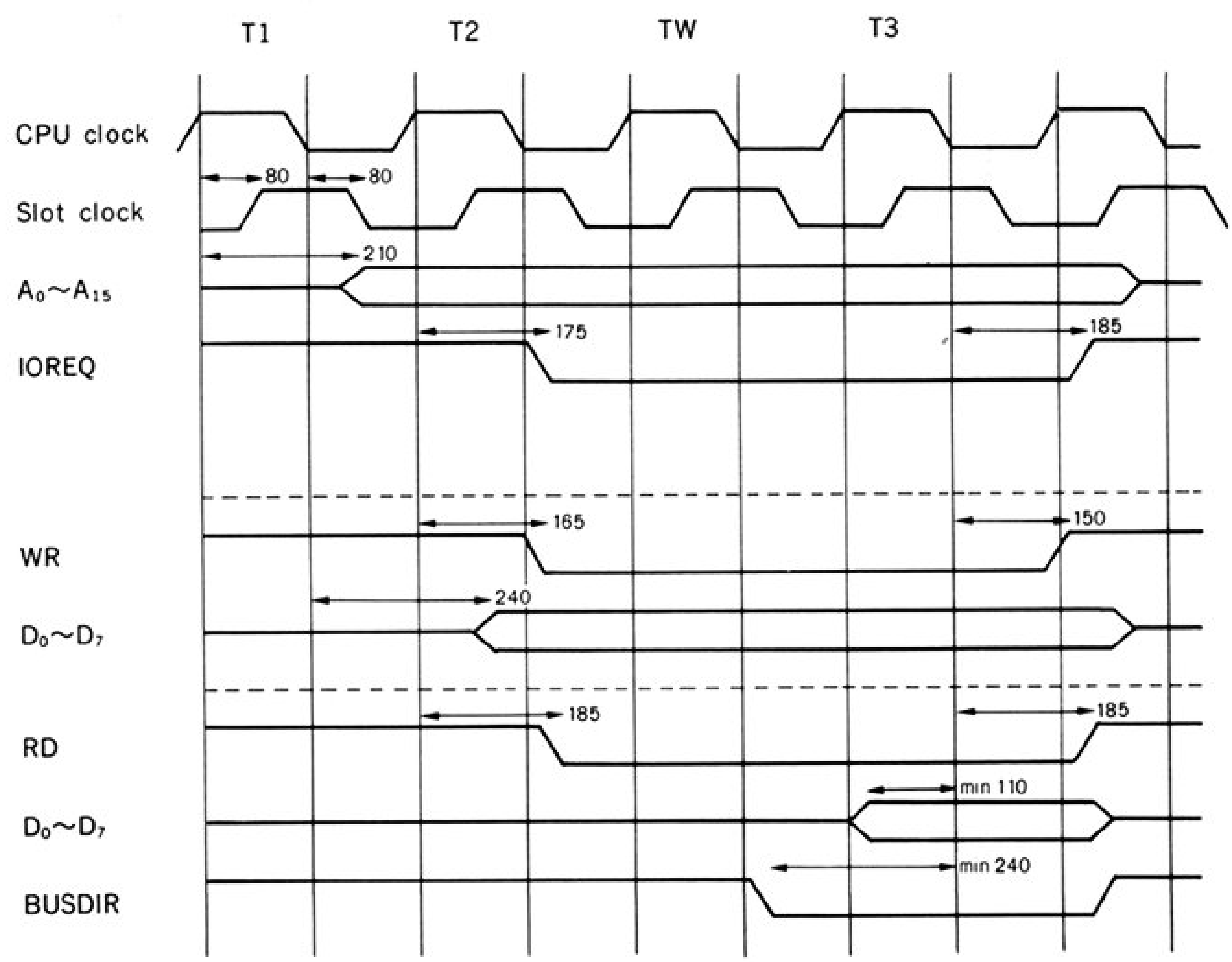
メモリサイクル 拡張スロット



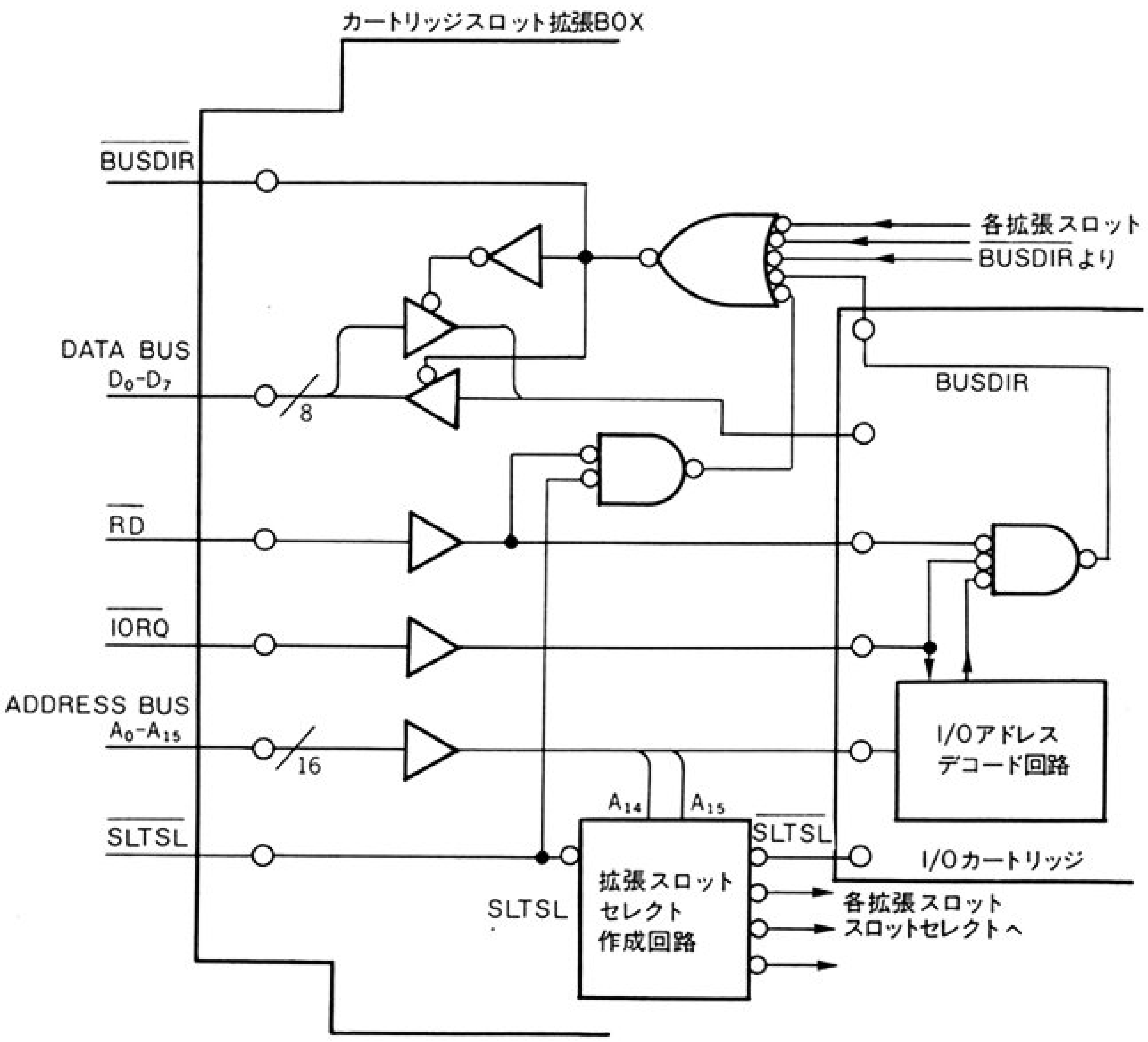
I/Oサイクル 基本スロット

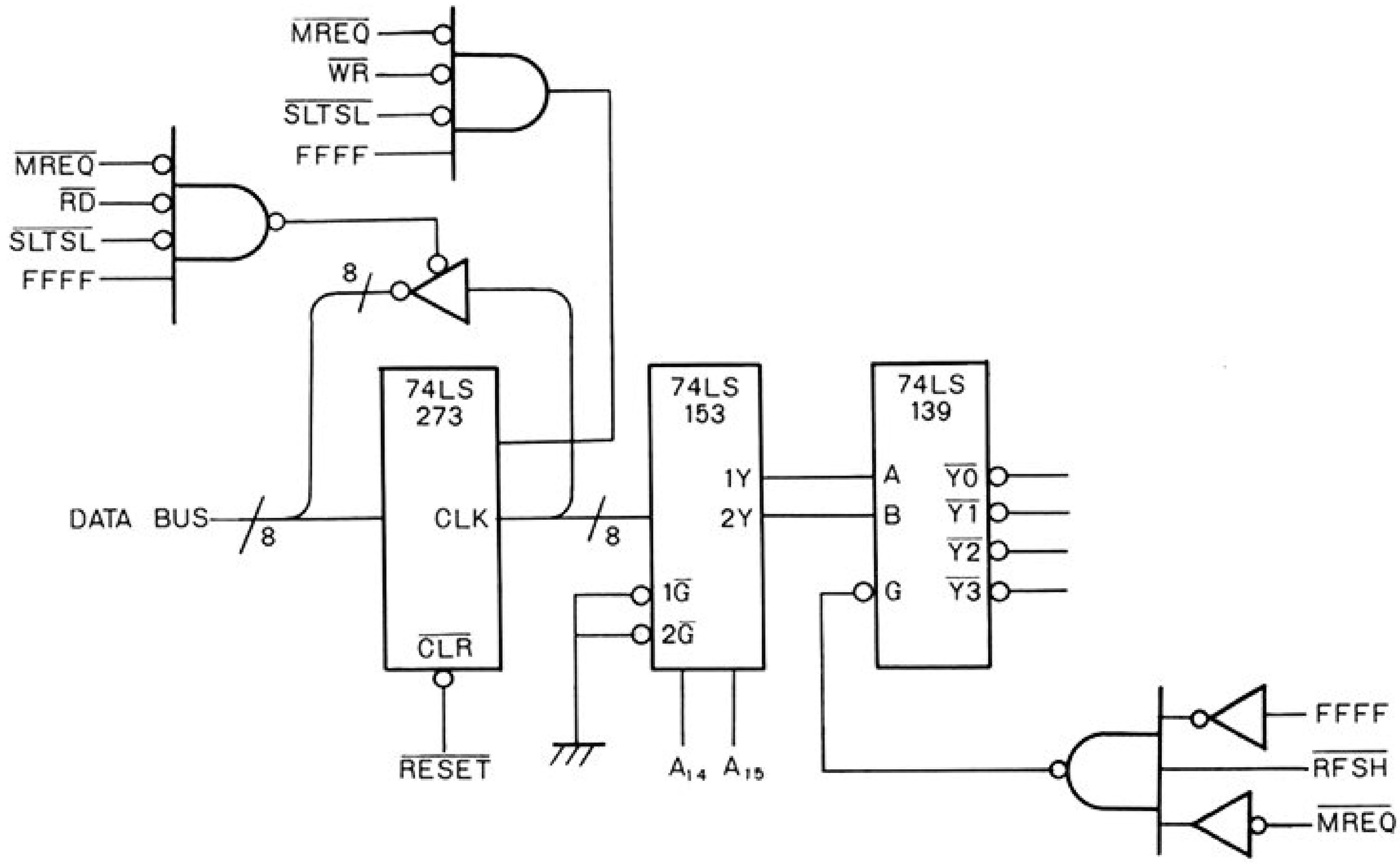


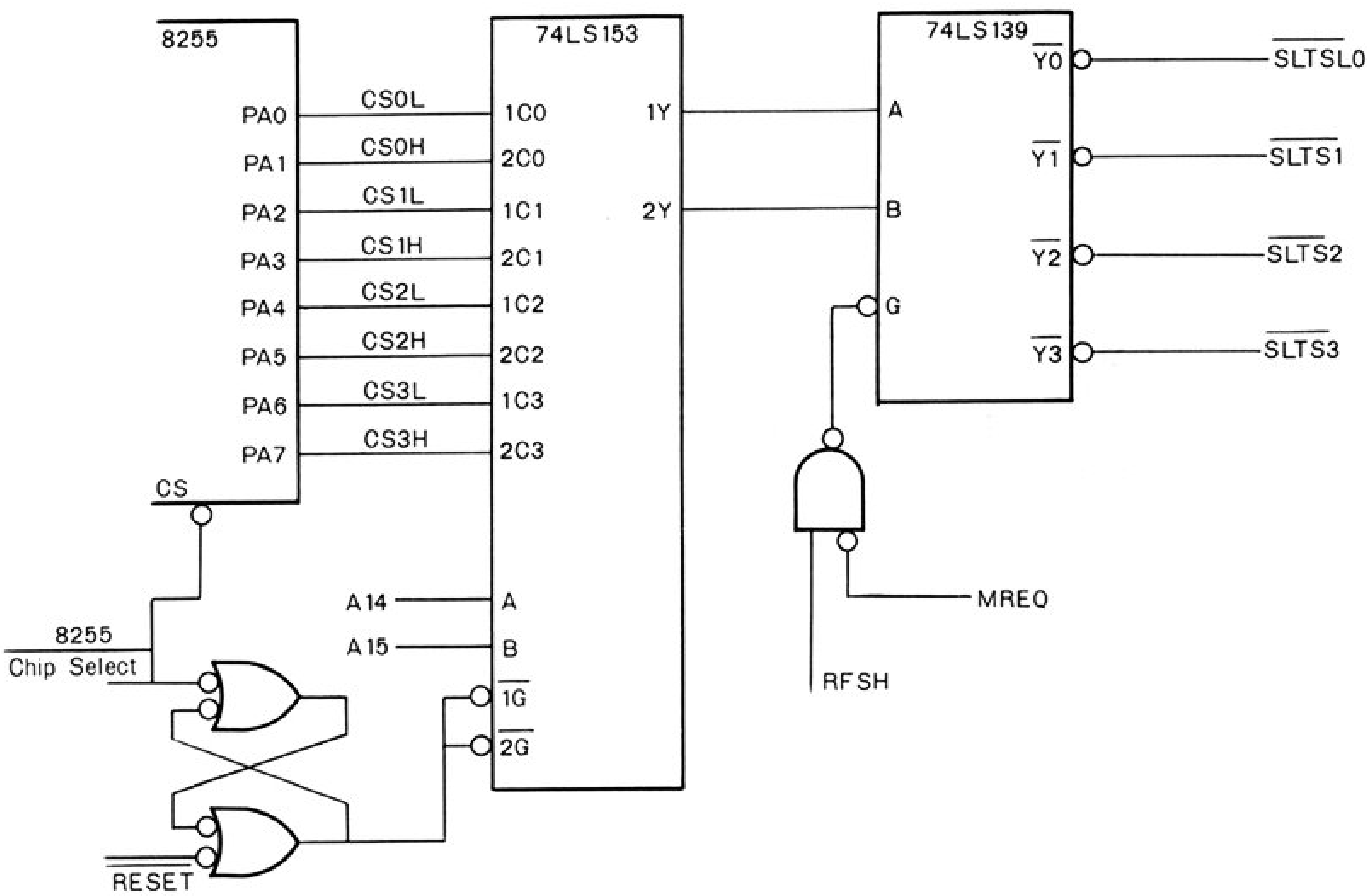
I/Oサイクル 拡張スロット



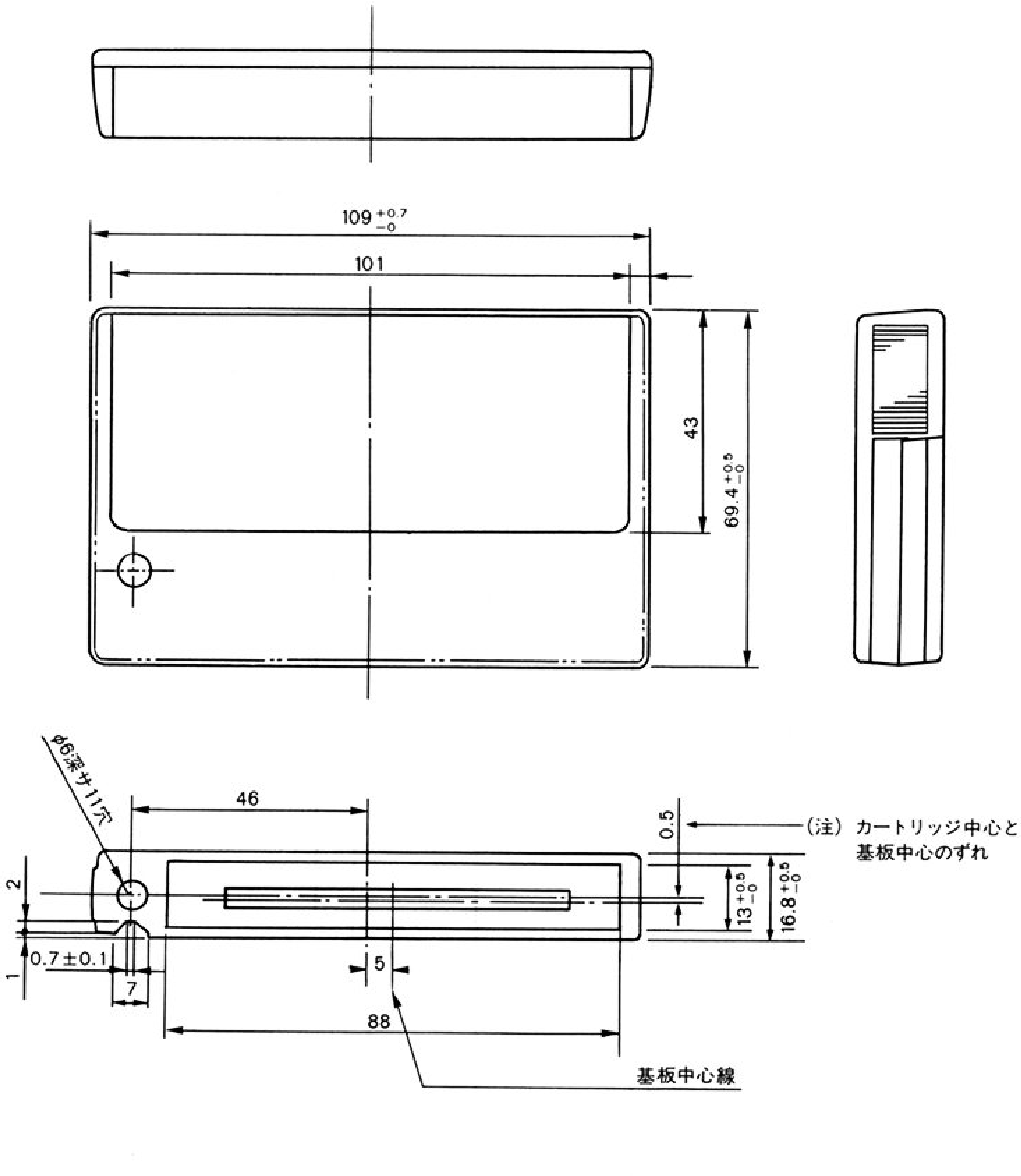
●拡張スロットセレクト信号作成回路例

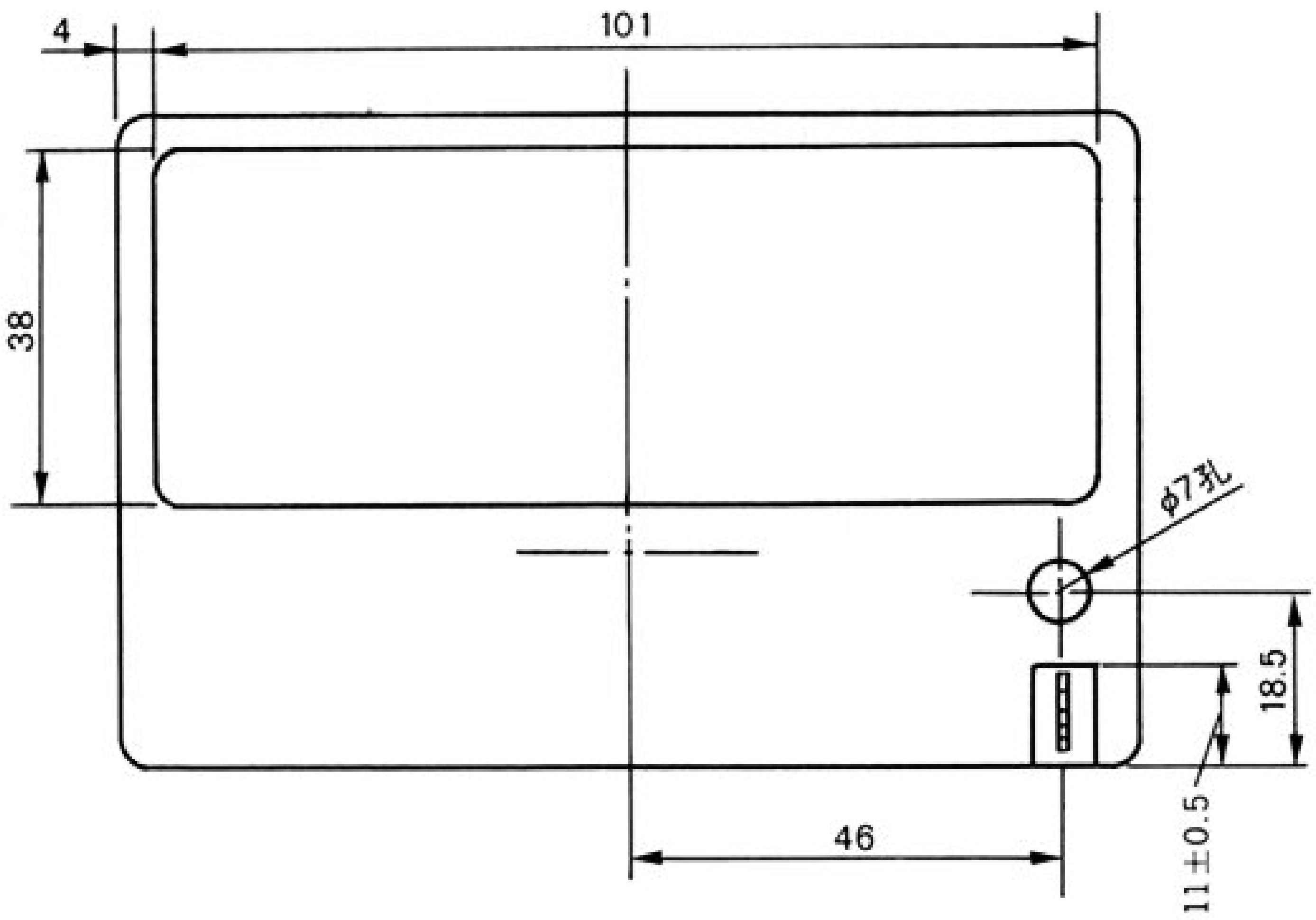



MREQ
WR
SLTSL
FFFF
MREQ
RD
SLTSL
FFFF
8
74LS
273
CLK
CLR
RESET
DATA BUS
8
8
74LS
153
1Y
2Y
1G
2G
A14
A15
74LS
139
A
B
G
Y0
Y1
Y2
Y3
FFFF
RFSH
MREQ

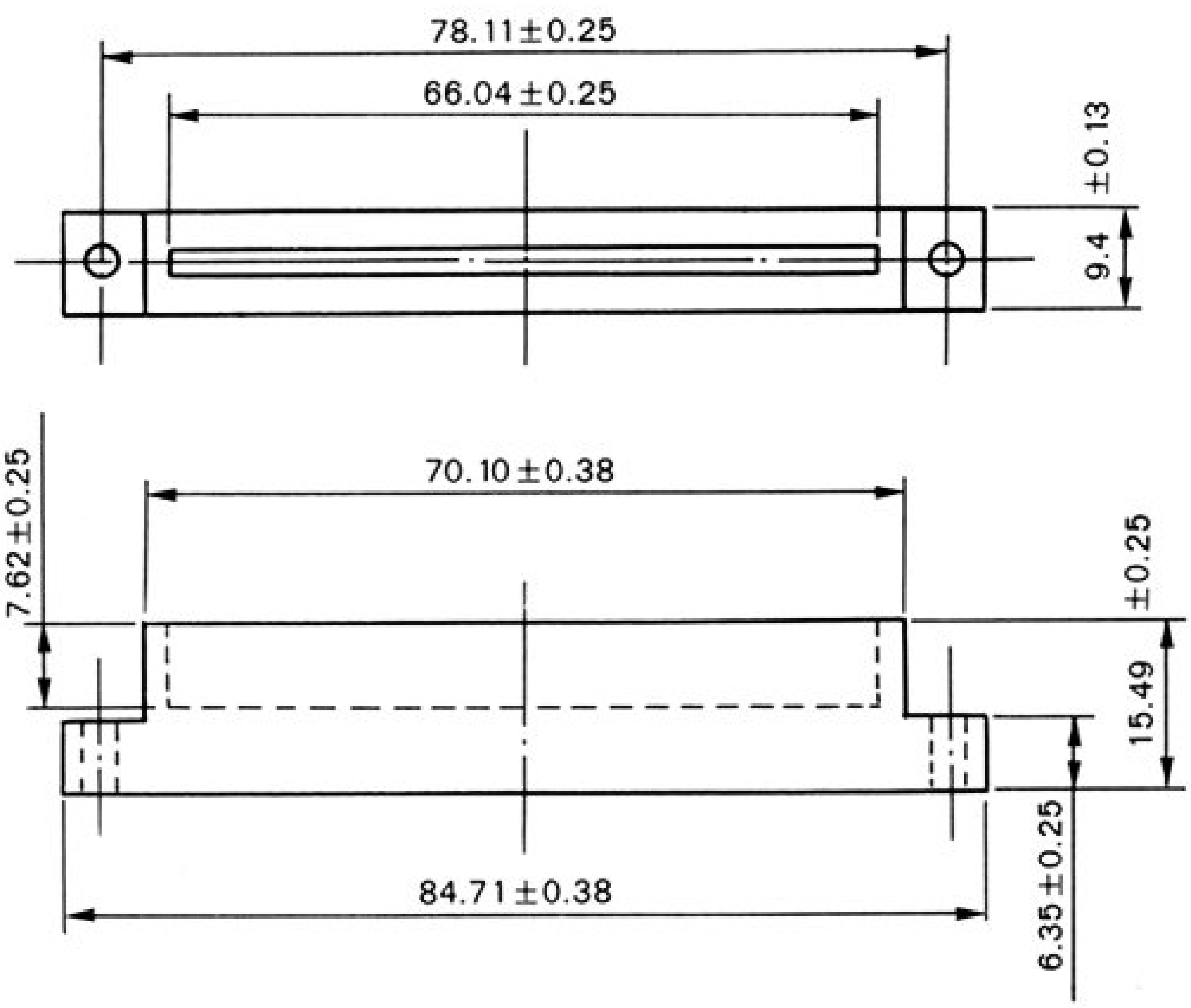

8255
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
CS
CS0L
CS0H
CS1L
CS1H
CS2L
CS2H
CS3L
CS3H
74LS153
1C0
2C0
1C1
2C1
1C2
2C2
1C3
2C3
1Y
2Y
A14
A
A15
B
1G
2G
8255
Chip Select
RESET
74LS139
Y0
Y1
Y2
Y3
A
B
G
SLTSL0
SLTS1
SLTS2
SLTS3
MREQ
RFSH

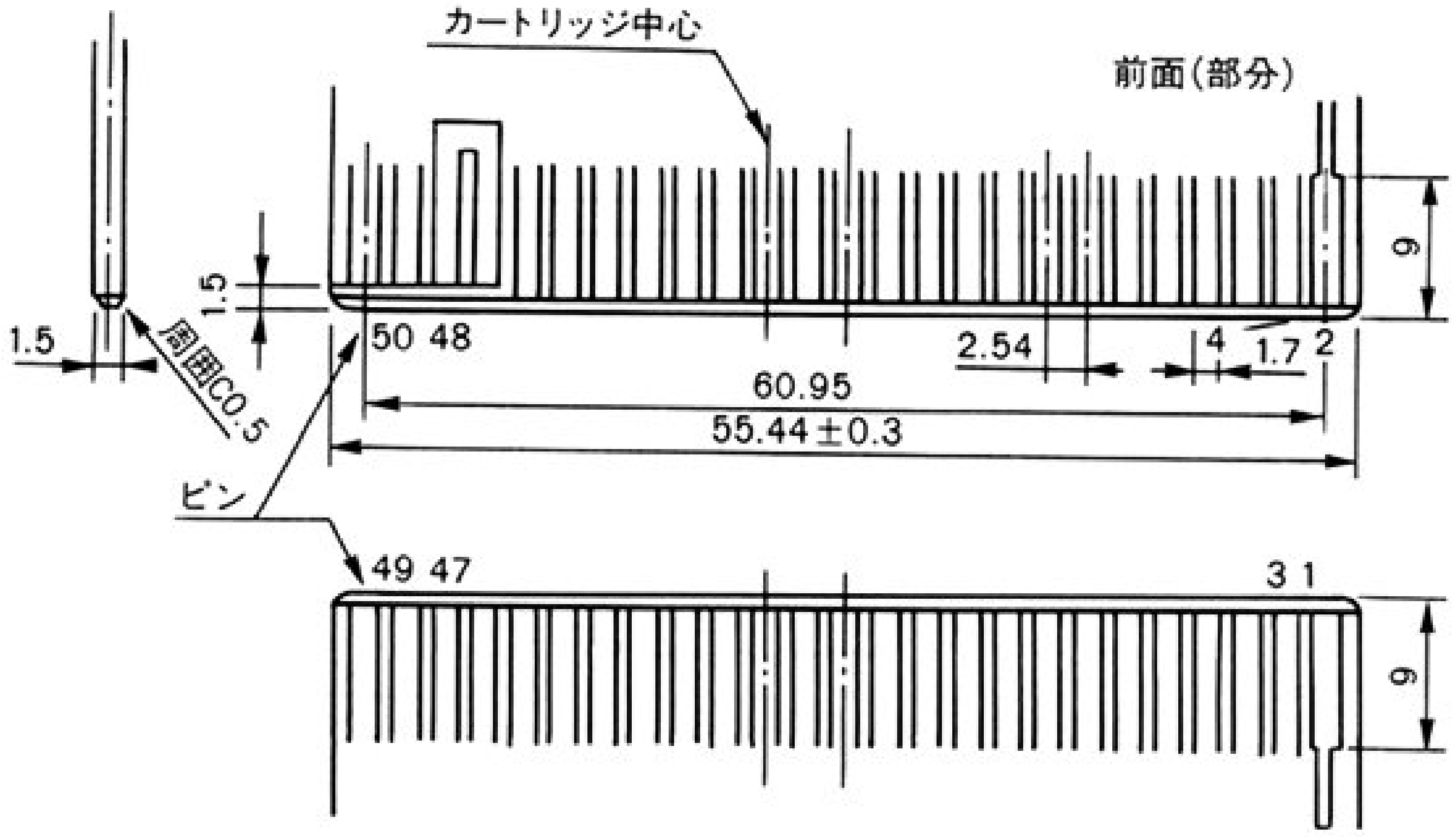
● カートリッジ寸法


109 +0.7 −0
101
43
69.4 +0.5 −0
φ6深サ11穴
46
0.5
(注) カートリッジ中心と
基板中心のずれ
2
13 +0.5 −0
16.8 +0.5 −0
0.7±0.1
1
7
5
88
基板中心線


4
101
38
φ7孔
18.5
11±0.5
46

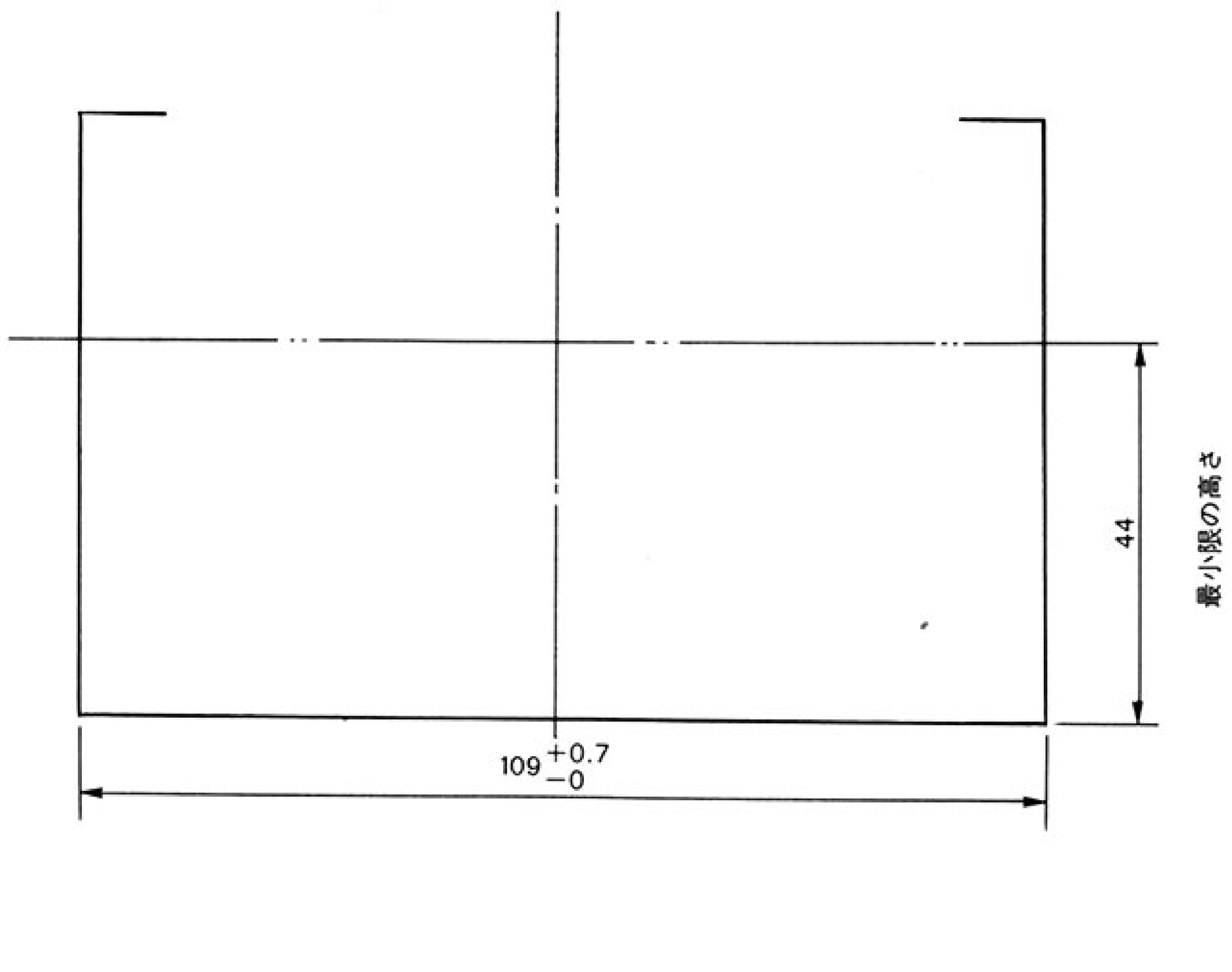
● コネクター寸法例


78.11±0.25
66.04±0.25
±0.13
9.4
70.10±0.38
7.62±0.25
±0.25
15.49
6.35±0.25
84.71±0.38



コネクタ（例） TI(carp)H4211∗∗−25
∗4selectable

注）カートリッジのコネクタ側の端から
44mm以内の部分に突起（スイッチ,
コネクタ等）を付けてはいけません.



A.8 コントロールコード表

コード（10進）	コード（16進）	機　　能	対応キー
0	00H		CTRL + @
1	01H	グラフィックキャラクタの入出力時のヘッダ	CTRL + A
2	02H	カーソルを直前の語の先頭へ移動	CTRL + B
3	03H	入力待ち状態を終了する	CTRL + C
4	04H		CTRL + D
5	05H	カーソル以下を削除	CTRL + E
6	06H	カーソルを次の語の先頭へ移動	CTRL + F
7	07H	スピーカを鳴らす（BEEP文と同じ）	CTRL + G
8	08H	カーソルの一つ前の文字を削除する	CTRL + H または BS
9	09H	次の水平タブ位置へ移動	CTRL + I または TAB
10	0AH	行送り（ラインフィード）	CTRL + J
11	0BH	カーソルをホームポジション（左上）に戻す	CTRL + K または HOME
12	0CH	画面をクリアし，カーソルをホームポジションに戻す	CTRL + L または CLS
13	0DH	カーソルを左端に戻す（キャリッジリターン）	CTRL + M または RETURN
14	0EH	カーソルを行末へ移動	CTRL + N
15	0FH		CTRL + O
16	10H		CTRL + P
17	11H		CTRL + Q
18	12H	挿入モードのON／OFFスイッチ	CTRL + R または INS
19	13H		CTRL + S
20	14H		CTRL + T
21	15H	1行を画面から削除	CTRL + U
22	16H		CTRL + V
23	17H		CTRL + W
24	18H		CTRL + X または SELECT
25	19H		CTRL + Y
26	1AH		CTRL + Z
27	1BH		CTRL + [または ESC
28	1CH	カーソルを右へ移動	CTRL + ¥ または →
29	1DH	カーソルを左へ移動	CTRL +] または ←
30	1EH	カーソルを上へ移動	CTRL + ^ または ↑
31	1FH	カーソルを下へ移動	CTRL + _ または ↓
127	7FH	カーソルの指す文字を削除	DEL

A.9 キャラクタコード表

コード(10進)	コード(16進)	キャラクタ	コード(10進)	コード(16進)	キャラクタ	コード(10進)	コード(16進)	キャラクタ	コード(10進)	コード(16進)	キャラクタ
0	00H	↑	32	20H	空白	64	40H	@	96	60H	`
1	01H		33	21H	!	65	41H	A	97	61H	a
2	02H		34	22H	"	66	42H	B	98	62H	b
3	03H		35	23H	#	67	43H	C	99	63H	c
4	04H		36	24H	$	68	44H	D	100	64H	d
5	05H		37	25H	%	69	45H	E	101	65H	e
6	06H		38	26H	&	70	46H	F	102	66H	f
7	07H		39	27H	'	71	47H	G	103	67H	g
8	08H		40	28H	(	72	48H	H	104	68H	h
9	09H		41	29H	)	73	49H	I	105	69H	i
10	0AH		42	2AH	*	74	4AH	J	106	6AH	j
11	0BH		43	2BH	+	75	4BH	K	107	6BH	k
12	0CH		44	2CH	,	76	4CH	L	108	6CH	l
13	0DH		45	2DH	−	77	4DH	M	109	6DH	m
14	0EH		46	2EH	.	78	4EH	N	110	6EH	n
15	0FH	コントロールキャラクタ	47	2FH	/	79	4FH	O	111	6FH	o
16	10H		48	30H	0	80	50H	P	112	70H	p
17	11H		49	31H	1	81	51H	Q	113	71H	q
18	12H		50	32H	2	82	52H	R	114	72H	r
19	13H		51	33H	3	83	53H	S	115	73H	s
20	14H		52	34H	4	84	54H	T	116	74H	t
21	15H		53	35H	5	85	55H	U	117	75H	u
22	16H		54	36H	6	86	56H	V	118	76H	v
23	17H		55	37H	7	87	57H	W	119	77H	w
24	18H		56	38H	8	88	58H	X	120	78H	x
25	19H		57	39H	9	89	59H	Y	121	79H	y
26	1AH		58	3AH	:	90	5AH	Z	122	7AH	z
27	1BH		59	3BH	;	91	5BH	[	123	7BH	{
28	1CH		60	3CH	<	92	5CH	¥	124	7CH	|
29	1DH		61	3DH	=	93	5DH	]	125	7DH	}
30	1EH		62	3EH	>	94	5EH	^	126	7EH	~
31	1FH	↓	63	3FH	?	95	5FH	_	127	7FH	削除

コード (10進)	コード (16進)	キャラクタ	コード (10進)	コード (16進)	キャラクタ	コード (10進)	コード (16進)	キャラクタ	コード (10進)	コード (16進)	キャラクタ
128	80H	♠	160	A0H		192	C0H	タ	224	E0H	た
129	81H	♥	161	A1H	。	193	C1H	チ	225	E1H	ち
130	82H	♣	162	A2H	「	194	C2H	ツ	226	E2H	つ
131	83H	♦	163	A3H	」	195	C3H	テ	227	E3H	て
132	84H	○	164	A4H	、	196	C4H	ト	228	E4H	と
133	85H	●	165	A5H	・	197	C5H	ナ	229	E5H	な
134	86H	を	166	A6H	ヲ	198	C6H	ニ	230	E6H	に
135	87H	ぁ	167	A7H	ァ	199	C7H	ヌ	231	E7H	ぬ
136	88H	ぃ	168	A8H	ィ	200	C8H	ネ	232	E8H	ね
137	89H	ぅ	169	A9H	ゥ	201	C9H	ノ	233	E9H	の
138	8AH	ぇ	170	AAH	ェ	202	CAH	ハ	234	EAH	は
139	8BH	ぉ	171	ABH	ォ	203	CBH	ヒ	235	EBH	ひ
140	8CH	ゃ	172	ACH	ャ	204	CCH	フ	236	ECH	ふ
141	8DH	ゅ	173	ADH	ュ	205	CDH	ヘ	237	EDH	へ
142	8EH	ょ	174	AEH	ョ	206	CEH	ホ	238	EEH	ほ
143	8FH	っ	175	AFH	ッ	207	CFH	マ	239	EFH	ま
144	90H		176	B0H	ー	208	D0H	ミ	240	F0H	み
145	91H	あ	177	B1H	ア	209	D1H	ム	241	F1H	む
146	92H	い	178	B2H	イ	210	D2H	メ	242	F2H	め
147	93H	う	179	B3H	ウ	211	D3H	モ	243	F3H	も
148	94H	え	180	B4H	エ	212	D4H	ヤ	244	F4H	や
149	95H	お	181	B5H	オ	213	D5H	ユ	245	F5H	ゆ
150	96H	か	182	B6H	カ	214	D6H	ヨ	246	F6H	よ
151	97H	き	183	B7H	キ	215	D7H	ラ	247	F7H	ら
152	98H	く	184	B8H	ク	216	D8H	リ	248	F8H	り
153	99H	け	185	B9H	ケ	217	D9H	ル	249	F9H	る
154	9AH	こ	186	BAH	コ	218	DAH	レ	250	FAH	れ
155	9BH	さ	187	BBH	サ	219	DBH	ロ	251	FBH	ろ
156	9CH	し	188	BCH	シ	220	DCH	ワ	252	FCH	わ
157	9DH	す	189	BDH	ス	221	DDH	ン	253	FDH	ん
158	9EH	せ	190	BEH	セ	222	DEH	゛	254	FEH	
159	9FH	そ	191	BFH	ソ	223	DFH	゜			

注）キャラクタコードのFFHはカーソル表示です．

● グラフィックキャラクタコード表

コード(10進)	コード(16進)	キャラクタ	コード(10進)	コード(16進)	キャラクタ
64	40H		80	50H	π
65	41H	月	81	51H	┴
66	42H	火	82	52H	┬
67	43H	水	83	53H	┤
68	44H	木	84	54H	├
69	45H	金	85	55H	┼
70	46H	土	86	56H	│
71	47H	日	87	57H	─
72	48H	年	88	58H	┌
73	49H	円	89	59H	┐
74	4AH	時	90	5AH	└
75	4BH	分	91	5BH	┘
76	4CH	秒	92	5CH	⊠
77	4DH	百	93	5DH	大
78	4EH	千	94	5EH	中
79	4FH	万	95	5FH	小

注）グラフィックキャラクタはすべて2バイト文字です．CHR$(1)を識別用のヘッダとし，続けて左記のコードを与えることによって表現します．

A.10 エスケープシーケンス表

○ カーソル移動	
<ESC>A	カーソルを上に移動
<ESC>B	カーソルを下に移動
<ESC>C	カーソルを右に移動
<ESC>D	カーソルを左に移動
<ESC>H	カーソルをホームポジションに移動
<ESC>Y<Y座標+20H><X座標+20H>	カーソルを（X，Y）の位置に移動
○ 編集，削除	
<ESC>j	画面をクリア
<ESC>E	画面をクリア
<ESC>K	行の終わりまで削除
<ESC>J	画面の終わりまで削除
<ESC>L	1行挿入
<ESC>M	1行削除
○ その他	
<ESC>x4	カーソルの形を'■'にする
<ESC>x5	カーソルを消す
<ESC>y4	カーソルの形を'_'にする
<ESC>y5	カーソルを表示する

索　引

執筆者紹介

鎗田　竜一

1964年３月生まれ．国際商科大学経済学部在学中より，㈱アスターインターナショナルにおいてMSXマシンの開発に携わる．

宮崎　暁

1963年12月生まれ．工学院大学機械工学科卒業．アスキーHSP(Home Software Products)において，「たわらくん」「サンダーボール」など，多くのMSX用ソフトの開発に携わる．

清水　真佐志

1963年10月生まれ．芝浦工業大学２部電気工学科在学中．鎗田氏とともにMSXマシンの開発に携わる．画像処理に興味を持ち，マルチMSX-VIDEOシステムには大きな関心を寄せている．

参考文献

- MSXテクニカルデータブック1［増補改訂版］
  アスキー・マイクロソフトFE本部編著　アスキー発行
- MSXテクニカルデータブック2
  アスキー・マイクロソフトFE本部編著　アスキー発行
- V9938 MSX-VIDEOテクニカルデータブック
  アスキー・マイクロソフトFE本部／日本楽器製造株式会社編　アスキー発行
- 応用CP／M
  村瀬　康治著　アスキー発行

MSX2 テクニカル・ハンドブック

1986年4月5日　初版発行
1990年4月1日　第1版第11刷発行
定価3,610円(本体3,505円)

監　修　アスキー・マイクロソフトFE
発行者　塚本慶一郎
発行所　株式会社アスキー
〒107 東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)486-7111(大代表)
情報TEL (03)498-0299(ダイヤルイン)
出版営業部TEL (03)486-1977(ダイヤルイン)

編集担当　桜田幸嗣
CTS　福田工芸株式会社
印刷　株式会社加藤文明社印刷所

ISBN4-87148-194-8

●11123／1112301

内容一覧

- ハードウェア概要
- ソフトウェア構成
- MSX BASIC ver. 2.0
- MSX-DOS
- VDPと画面表示
- PSGと音声出力
- カセット・インターフェイス
- キーボード・インターフェイス
- プリンタ・インターフェイス
- 汎用入出力インターフェイス
- CLOCKとバッテリバックアップ・メモリ
- Appendix 各種仕様一覧

ISBN4-87148-194-8 C3055 P3610E

定価3,610円(本体3,505円)